AFIT/GAM/ENC/99D-01

A NUMERICAL SIMULATION OF A CARBON BLACK SUSPENSION CELL VIA A TIME–REVERSED, DOUBLE LAYER COMPUTE ALGORITHM

THESIS

Gregg Thomas Anderson

AFIT/GAM/ENC/99D-01

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/GAM/ENC/99D-01

A NUMERICAL SIMULATION OF A CARBON BLACK SUSPENSION CELL VIA A TIME–REVERSED, DOUBLE LAYER COMPUTE ALGORITHM

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Applied Mathematics

Gregg Thomas Anderson, B.S. Ed, B.S. EE

December, 1999

AFIT/GAM/ENC/99D-01

A NUMERICAL SIMULATION OF A CARBON BLACK SUSPENSION CELL VIA A TIME–REVERSED, DOUBLE LAYER COMPUTE ALGORITHM

Gregg Thomas Anderson, B.S. Ed, B.S. EE

Approved:

Dr. William P. Baker Thesis Advisor	Date
Dr. Dennis W. Quinn Committee Member	Date
Dr. Roger J. Becker Committee Member	Date
Dr. Daniel G. McLean Committee Member	Date

Acknowledgements

I would like to express my sincere appreciation to my faculty advisor, Dr. William Baker and my colleague Dr. Roger Becker, for their guidance and support throughout the course of this thesis project. I am particularly grateful to Dr. Becker for his patient and thoughtful answers to my repeated questions. His insight and experience have been invaluable. I am also grateful to Dr. Dan McLean for his many insightful comments and to Dr. Ruth Pachter for her helpful suggestions. My thanks also to the "Unix God", Dr. James Lupo, whose knowledge of Unix and computers rescued me more than once. What is Ohio's loss will be Arizona's gain.

I would also like to thank my former supervisor, Ms. Pamela Schaefer of the Hardened Materials Branch, Materials and Manufacturing Directorate, for her encouragement, support and generous latitude given me throughout these many months. Now the real work begins.

But most of all, I would like to thank my wife, Deb, for enduring all the lonely nights and lost weekends to see this project through. Without her loyal support, her patient sacrifice and unwavering love, this journey of mine would have been hopeless. Thank you, sweetheart.

Gregg Thomas Anderson

Table of Contents

Page

Acknowledgements iii

List of Figures vi

List of Tables x

Abstract xi

I. Introduction 1-1

1.1 Rationale 1-1

1.2 Computer Simulation 1-2

II. Physics 2-1

2.1 Essential Parameters 2-1

2.2 Assumptions 2-4

2.3 Electron Collision Frequency and Diffusion Constant 2-5

2.4 Light Scattering 2-7

III. Code Description 3-1

3.1 Previous Code 3-1

3.2 ZAP 3-2

3.3 Present Code 3-5

IV. The Algorithm and Its Results 4-1

4.1 The Computational Scheme 4-1

4.2 Results 4-8

V. Conclusion 5-1

Appendix A. The Saha Equation A-1

Page

Appendix B. The Timestep Calculation B-1

B.1 Factors Influencing Choice of Time Step B-1

B.2 Estimation of Initial Time Step B-2

Appendix C. Partial Code Listing C-1

Bibliography BIB-1

Vita VITA-1

List of Figures

Figure — Page

1.1. A shadowgraph of the Primary Zone formed by a laser entering a liquid medium. Note the bubbles fanning out from the Primary Zone. 1-4

1.2. Schematic of the shadowgraph of Figure 1.1. The dark arrows point in the general direction of the laser pulse toward a focal point and represent the boundary of the Primary Zone. The white arrows represent the general direction of bubble formation as they fill the volume which surrounds the Primary Zone. 1-4

1.3. Input arrangement assumed by the simulation. A positive lens focuses the laser into a sample of thickness L. Near the exit surface, the plasma ignites in the focal region, starting the simulation which tracks the progress of plasma ignition toward the entrance surface. The secondary zone receives scattered light from the Primary Zone. 1-6

1.4. The leftward advance of the plasma in the upper half of the Primary Zone of Figure 1.3. The first layer to ignite is nearest the exit surface in part (a). Layer 2 remains unignited because it's threshold fluence level has not been reached. Soon, layer 2 ignites in part (b) and similarly for layer 3 in part (c). 1-7

1.5. Three-dimensional focusing volume of the Primary Zone. The 1D simulation produces intensity values at the center of the focal region's exiting disk, i.e. at the tip of the arrow. 1-8

1.6. An event A in the sixth layer must be carefully tracked through five intervening layers, each with overlapping nonlinear time grids, to propagate correctly to B. The output from layer 6 becomes the input to layer 5 and so on. 1-9

1.7. After the first 100mJ/cm^2 of fluence, layer 1 activates and processes information directly to output until Tau(2). From Tau(2), the program must process output from layer 2 through layer 1 to output. Layer 3 must process output through layer 2 and layer 1, etc. . . . 1-10

Figure Page

2.1. Coordinate geometry of the incident electric field in the Z direction and the scattering direction, r, for Rayleigh scattering. The particle with radius a is positioned at the origin. (After Kerker [5: page 33]). 2-8

3.1. Grid geometry of the Primary Zone for the new algorithm. J denotes layer number and HH the inter-layer distance. Rad is the radius for a given layer. The base of the figure denotes the Z axis. For the 1D case, output is calculated at the rightmost point of the Z axis at J = 1. 3-2

3.2. Flowchart for the subroutine ZAP. 3-3

3.3. The J+1 layer of values is calculated first. Next, layer J starts. But the nonlinear time axis of layer J is offset from that of layer J+1. As a result, some of the K time bins will be contained in a KQ bin and others will fall on a boundary and must be divided. Hence, a portion of the K+1 bin (XX) must be sent to KQ and a portion (YY) to KQ+1. 3-6

3.4. This figure illustrates conceptually how the first three layer "histories" overlap and how each successive layer starts earlier in the pulse. The start time for layer 20 is Tau20 and, for its lifetime, processes "free" input only. At the same time the processing for layer 20 loads the input aray for layer 19. The procedure starts over again when layer 19 starts at Tau19. But layer 19 processes "free" input only until Tau20. From Tau20 to completion, layer 19 takes its input from the array loaded by layer 20. 3-8

3.5. Schematic representation of a hand-off. Part (a) represents a conceptual time bin and part (b) shows an output variable calculated for that time bin. Part (c) shows the output variable being assigned to the input time bin on the left. Part (d) displays this pattern of assignment for a series of time bins comprising layer J which is the active layer starting at the bottom and progressing toward the top. The output of layer J awaits layer J-1 when layer J-1 becomes active. 3-10

Figure Page

3.6. The first layer (layer 20) is a special case. It receives only uninterrupted laser energy. After passing the first 100 mJ/cm^2, the layer ignites. The layer then steps through its time bins, loading its output array which yields a plot of the output which is input for the next cycle. 3-11

3.7. Layer 19 starts out receiving free input but soon runs into layer 20. Layer 20's array now becomes the input to layer 19 which loads its arrays. When plotted, they yield an attenuated curve (displayed darker for clarity). Once layer 19 is complete, layer 20 can be ignored for the rest of the pulse. 3-11

4.1. Flowchart for the main program. 4-2

4.2. Each box in the bottom row represents an input value of direct laser energy. Those boxes filled with an X contain calculated values; those without are waiting to be calculated. As K steps through time from left to right, ZAP attenuates each input and assigns it to an output box on the top to be processed later by layer 19. 4-4

4.3. After the offset covered by loop B, loop C must now process input previously loaded by loop A. 4-5

4.4. Conceptual positions of the A, B and C loops as they pertain to layers 19 and 20. Once layer 20 is computed, layer 19 and the rest of the layers only experience loops B and C. 4-7

4.5. The input to the simulation is gaussian instead of a Q-switched pulse shape due to an external subroutine which uses the readily available formula for the Gaussian distribution. 4-8

4.6. The output of the older version of the simulation. Note: in each plot profile, the timeline for the in–coming pulse extends to 28 nsecs. . . 4-9

4.7. The output of the newer version of the simulation. 4-10

4.8. Both the old and new outputs are plotted together. Note the close similarity of both plots. 4-10

4.9. Output intensity as a function of pulse energy for the older version. 4-11

4.10. The output intensity of the new version as a function of pulse energy. 4-12

Figure Page

4.11. Old output as a function of plasma radius, A0. The bottom curve is for A0 = 8.0E-06 cm and is included as a baseline for comparison with the curves above it generated from the smaller values of A0. In the legend, 800 is 8.0E-06 cm, 190 is 1.90E-06 cm and 170 is 1.70E-06 cm. 4-12

4.12. New output as a function of plasma radius, A0. The bottom curve is again for A0 = 8.0E-06 cm with the same legend designation as in Figure 4.11. 4-13

4.13. The effects of selected layers on a single pulse are shown superimposed for layers 19 through layer 2. In the legend, xy in hhhxy refers to the layer number. The curves closer to the origin are layers nearer to focus. 4-13

5.1. Off-axis scattering will be accomplished through an array of node points roughly spanning a cross-section of the Primary Zone. Only the upper half need be used because of symmetry. The bottom line is the present z-axis. 5-3

5.2. Part (a) displays three shaded grid points which supply input to the single unshaded neighbor grid point. The symmetric input/output scheme is shown in terms of grid position in part (b). 5-4

5.3. The cross-sectional plane of grid points, when rotated around the z-axis, will span the roughly cone-shaped Primary Zone volume. . . . 5-4

B.1. The sharp rise time of the input Q-switched laser pulse dictates a nonlinear time partition at the beginning to capture and sustain the critical physics of scattering and absorption. B-2

B.2. In the first time step, the plasma expands from r to $r + \Delta r$. B-3

List of Tables

Table — Page

3.1. Timing data representative of typical run times for both the old and new versions of the code. "User" is the time spent executing the code. "System" is the time spent performing system services such as accessing the file system, reporting time of day, etc. 3-7

4.1. The input data set used for both the old and new codes. Gamma is the ratio of the heat capacity at constant volume to the heat capacity at constant pressure. F-number is the ratio of the focal length of a lens to its diameter. 4-9

5.1. The range of input variables determining the validity of the present code. 5-2

Abstract

A numerical simulation of a carbon black suspension cell is explored which models a laser–induced plasma within a liquid ethanol medium of approximately 1mm thickness. The simulation model assumes a laser pulse with a pulse width of approxmiately 9 nsecs propagating in the left–to–right direction striking the front surface of the medium and focusing to a spot within the liquid volume. When the energy density within a given irradiated volume is sufficiently high, it ignites the carbon particles and generates a large number of free electrons, i.e. a plasma. The plasma couples with the in–coming laser energy on a picosecond timescale, thereby attenuating the intensity of the remaining pulse as it traverses the medium. The simulation divides the sample into discrete layers and models the propagation of the plasma through the course of a single laser pulse containing a total fluence of 1 Joule/cm^2. A new double layer, time–reversed algorithm is employed which modifies and extends the capabilities of the existing code. The older version is used as a baseline for comparison with the new program.

A NUMERICAL SIMULATION OF A CARBON BLACK SUSPENSION CELL VIA A TIME–REVERSED, DOUBLE LAYER COMPUTE ALGORITHM

I. Introduction

1.1 Rationale

The aim of this thesis is to design a computer code which will aid in understanding the physical interaction of an intense beam of light with a suspension of absorbing particles in a liquid. An examination of the data in experiments on such a system indicates physical effects which cannot be accounted for by conventional descriptions of the system response. To remedy this deficiency, researchers developed a theory or model of the response which incorporates assumptions subject to challenge. The system response is very complicated, so that comparisons with suitably designed experiments require the incorporation of the model into a numerical code. A previous effort was carried out with this aim. The resulting code, which I designate as the old code, suffered from two limitations. The first is that it used a one–dimensional desciption of the response. Therefore, it could not account for scattering in the radial direction. This thesis describes the precursor to a new two–dimensional code. The precursor overcomes this defect by laying the groundwork for the future modeling of scattering effects. The second problem with the old code was that it was not amenable to modeling elaborate physical descriptions. Since a multi–level description of the processes which occur in the response places a heavy burden on computational time, the program described in this thesis was written so that, with appropriate modifications, it can run on a computer with multiple processors using parallel programming techniques. The extant software was updated to run efficiently on an IBM SP2 machine.

This thesis describes the physical assumptions on which the model rests. However, it does not revalidate those assumptions. It concentrates on modifying the old code so that later it can test those assumptions by observing how well the code predicts laboratory results. If, after the code has been thoroughly tested to confirm that it is correctly

implementing the model, the code fails to be an accurate predictor of actual events, the physical assumptions of the model will have to be modified.

The old code, and the present code which has been derived from it, form a small part in a much more vast computational mosaic. Materials researchers use an extensive array of modeling software to analyze and predict material and device characteristics in an effort to study and predict how well materials interact with light. Materials modeling techniques employ *ab initio* first principles or semi–empirical approaches. The aim is to elucidate the physical mechanisms at the molecular level without having to resort to costly and time consuming experiments. In some cases, such simulated experiments are the only way to gain insight into the underlying behavior. The magnitude and complexity of the present investigations require that researchers must routinely rely on the computational resources provided by the Major Shared Resource Center (MSRC) located here at Wright–Patterson AFB and at other sites across the DoD.

1.2 Computer Simulation

The code developed for this thesis models the interaction of a single pulse of laser energy with a thin, liquid cell. The code, however, can be generalized to describe other systems. The geometry involved is, of course, three–dimensional. The simulation, however, selects certain 2–D regions or "slices" of the cell where the interaction takes place and processes these as representative of the phenomenon in two–dimensions. Then, to fill an appropriate 3–D volume, the slice is rotated around an axis through the center of the cell. If the slice spans a diameter, the rotation is through π radians. If the slice spans a radius, the rotation is through 2π radians.

The material medium used in this study is composed of carbon micro–particles suspended in ethanol, of a pre–determined thickness, and positioned at right angles to the direction of propagation of the laser pulse. The computer program divides the medium into discrete layers within a cone–shaped region called the Primary Zone (PZ). This is the volume filled by the laser beam traveling from left to right as it narrows to a focal spot. Dividing the Primary Zone this way captures the ignition of the carbon particles within it which starts the plasma and, thereafter, captures the physical mechanisms of absorption

from electrons and scattering from bubbles as the plasma progresses from layer to layer. This not only allows a numerical "divide and conquer" approach but it also affords a means of tracking the physical progress as well as the accumulating effects of the ignited plasma. A typical image, or shadowgraph, of the different regions within the sample is shown in Figure 1.1. The Primary Zone is the cone–shaped, dark region and is the volume directly illuminated by the laser beam. It contains the plasma generated by the ignition of the carbon particles within it. Another region, however, surrounds the Primary Zone. This more spherical secondary zone is populated predominantly by bubbles which, as experimental evidence indicates, scatter the laser radiation. Evidently, enough light is being scattered out of the Primary Zone with sufficient energy to create these secondary zone bubbles. The image in Figure 1.1 shows the effects of a single laser pulse entering a liquid medium from the left and narrowing to focus at the end of a dark, cone-shaped region—the Primary Zone. The shadowgraph image, of course, is a two–dimensional view. A spatial schematic of the regions involved is shown in Figure 1.2.

The shadowgraph reveals the accumulation of a number of events occurring over the lifetime of a single pulse. Based on the most current analysis (see Goedert [3]), at the beginning of the pulse, bubbles and hot electrons are generated within the Primary Zone. But enough energy is scattered out of the Primary Zone that bubbles are created a short time later in the secondary zone. Toward the end of the pulse, laser energy scatters from the Primary Zone and off the secondary zone bubbles with the result that the laser pulse scatters from a much larger volume than just the Primary Zone alone. Unfortunately, current shadowgraphs of the kind shown in Figure 1.1 are not time–resolved so as to capture this sequence of events.

The geometry of the simulation is shown in Figure 1.3. Here, the slice of interest involves the upper half of the Primary Zone. For this thesis project, however, only the Z axis of the Primary Zone was investigated. The code is, therefore, a one–dimensional code. It is the goal of later work to model the contribution of the scattering effects from the secondary zone by modeling the upper slice of the Primary Zone and then rotating it 2π radians around the Z axis.

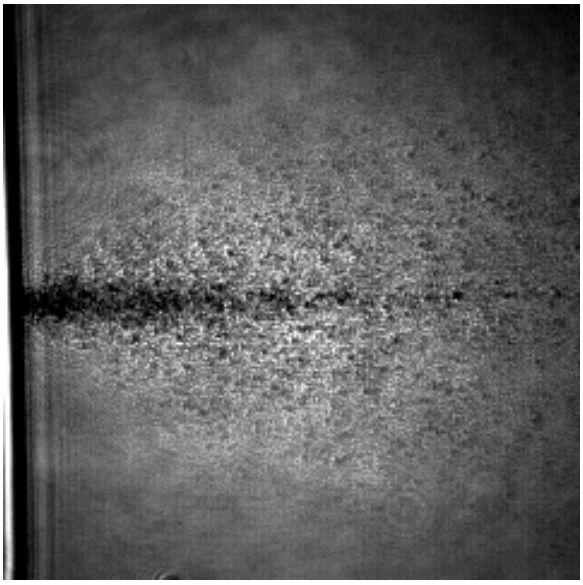

Figure 1.1 A shadowgraph of the Primary Zone formed by a laser entering a liquid medium. Note the bubbles fanning out from the Primary Zone.

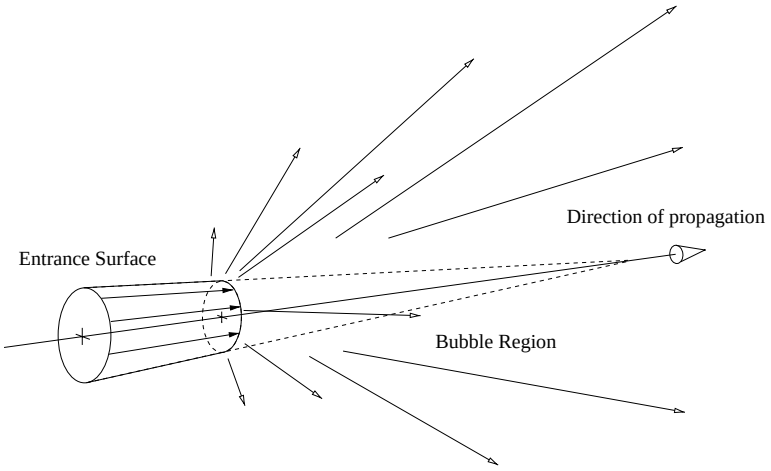

Figure 1.2 Schematic of the shadowgraph of Figure 1.1. The dark arrows point in the general direction of the laser pulse toward a focal point and represent the boundary of the Primary Zone. The white arrows represent the general direction of bubble formation as they fill the volume which surrounds the Primary Zone.

The code calculates the attenuating effects of a plasma and bubbles by first assuming a sample of thickness L irradiated by a laser pulse from a positive lens, as shown in Figure 1.3. The positive lens converges the beam toward the back (exit) surface of the sample. The code sets up the necessary space and time variables to track the progress of the plasma within the laser beam volume (the Primary Zone). These variables track the progress of the plasma over the lifetime of a single laser pulse. The beam enters the sample from the left already converging from the input lens and comes to focus in the first layer. As the beam narrows, the intensity (W/cm^2) increases since the same amount of light is being forced to cover a smaller and smaller area. After a small amount of time, the fluence (J/cm^2), which is the time integral of intensity, will become high enough to ignite its carbon particles, which starts the plasma. (See Figure 1.4). The second layer, upstream from the first, will ignite but only after it too has attained enough fluence in like manner. Similarly for the third layer and so on toward the entrance surface. Thus the timing of ignition of each successive layer is an approximation to the development of the plasma within the length of the sample. The resulting attenuation of the laser beam occurs because the ignition sequence of the layers' plasma advances toward the entrance surface, coupling head on with the incoming laser energy in front of it, reducing the beam's intensity as it progresses rightward, shielding the detector.

The previous version of the code calculates the attenuation of the laser from the plasma and bubbles by starting at the exit surface on the Z axis. The code then transfers those results to neighboring layers through an accumulating process toward the entrance surface (see Figure 1.4). The calculation starts the moment the carbon particles in the layer at the focal volume vaporize and is designated Tau(n) for layer n. It must be emphasized, however, that the plasma does not propagate like the spread of a flame front throughout the Primary Zone backwards from the first ignition point in the starting layer. Rather, the ignition of the plasma is dependent on the geometry of the Primary Zone, each layer igniting locally only after 100 mJ have passed that particular layer. In other words, if the edges of the Primary Zone were parallel, the entire volume would ignite simultaneously.

The previous version accumulates the attenuating effects of the newly formed plasma and bubbles by adding new layers through time. Each calculation is coordinated with a

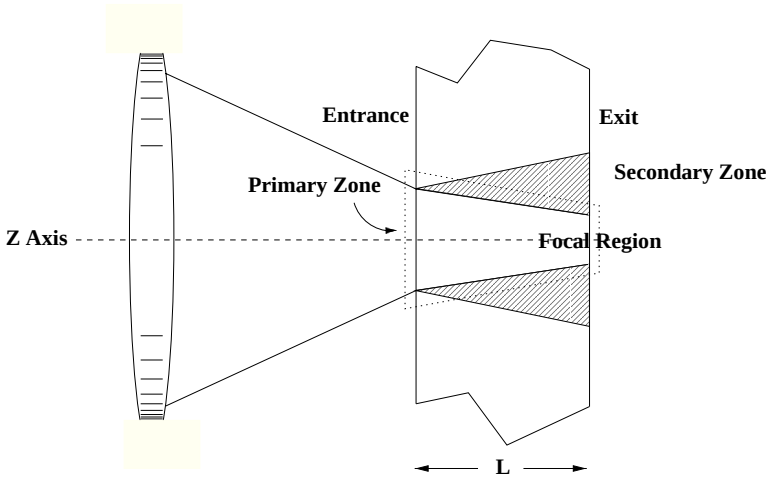

Figure 1.3 Input arrangement assumed by the simulation. A positive lens focuses the laser into a sample of thickness L. Near the exit surface, the plasma ignites in the focal region, starting the simulation which tracks the progress of plasma ignition toward the entrance surface. The secondary zone receives scattered light from the Primary Zone.

"wall clock" that acts as the master clock for the simulation of the entire pulse. The program starts at time $t = 0$ and progresses toward the end of the pulse, adding slices in a *leftward* progression through space since that is the direction of progress of the plasma. A three-dimensional view of the Primary Zone is shown in Figure 1.5. Here, the entrance surface is layer 20 and the exit surface is layer 1. The plasma ignites at the tip of the arrow and progresses rearward toward the entrance surface.

The f/# (f–number) of the lens is a key parameter for the simulation, and illustrates how changing it affects how the code adjusts individual layer ignition times. With large f/#'s, i.e. when the focal length of the lens is much greater than its diameter, each layer's ignition time will be nearly the same because each layer, which acts like a perpendicular slice through the cone, will have more nearly the same area and, therefore, the intensity difference between layers will not be great. Small f/#'s lead to the reverse the effect, resulting in a larger divergence of ignition thresholds among the layers. The effect of the former is a more instantaneous ignition of the plasma across the entire volume of the Primary Zone. The effect of the latter is to retard the plasma's leftward progress.

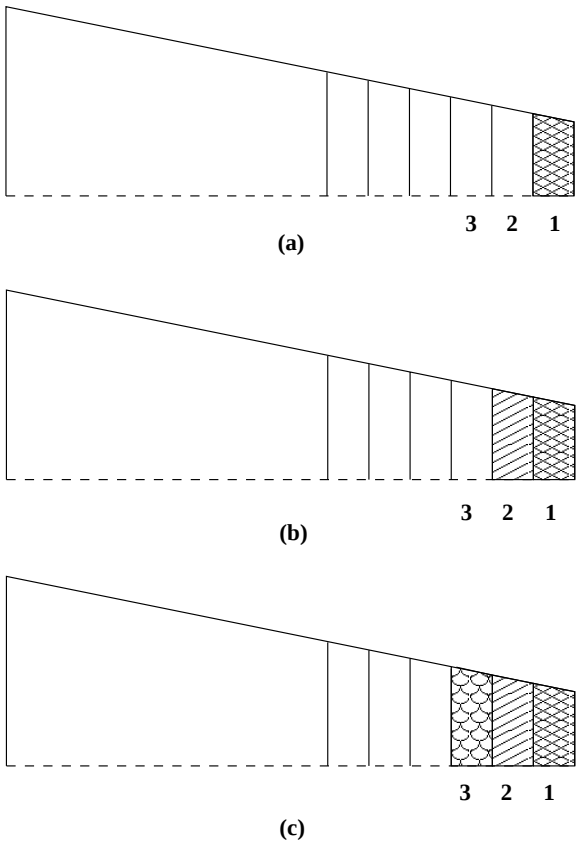

Figure 1.4 The leftward advance of the plasma in the upper half of the Primary Zone of Figure 1.3. The first layer to ignite is nearest the exit surface in part (a). Layer 2 remains unignited because it's threshold fluence level has not been reached. Soon, layer 2 ignites in part (b) and similarly for layer 3 in part (c).

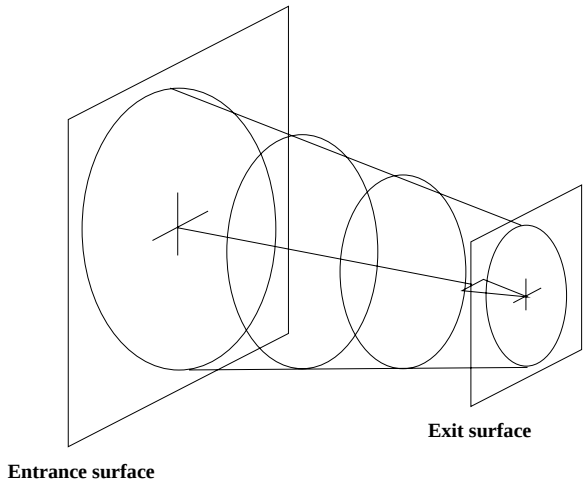

Figure 1.5 Three-dimensional focusing volume of the Primary Zone. The 1D simulation produces intensity values at the center of the focal region's exiting disk, i.e. at the tip of the arrow.

Since the simulation is one-dimensional, the results are determined at the z axis. Problems occur, however, as the number of layers increases. For the purposes of computational efficiency, each layer's time line is nonlinearly partitioned, i.e. each layer's time steps are clustered near its ignition point. Much of the critical response phenomenon occurs during plasma ignition and the subsequent rapid electron diffusion, which happens within roughly the first 600 picoseconds. As a result, the model must impose this high rate of energy injection so that the simulation corresponds to the evolving temporal dynamics. In a sense, energy must continually "stoke" the plasma to keep it sustained. Otherwise, with time steps too large, the electrons become starved of energy and quickly cool, extinguishing the plasma and its attenuating effects.

Hence, a nonlinear time scale must be imposed on each layer as shown in Figure 1.6. These time scales are represented as vertical fences, one for each layer, and "stacked" within themselves more closely together near the bottom than at the top. The code must resort to a complex logistical tracking system for managing the time and space variables for a given slice or layer. An event happening at A, for example, must have its attenuation "expressed" through five do-loops beginning with layer five. Layer five's history will be updated or "brought up to" the timeline associated with the arrow AB in Figure 1.6 by taking its input from its "leftward" neighbor and producing an output which must be made

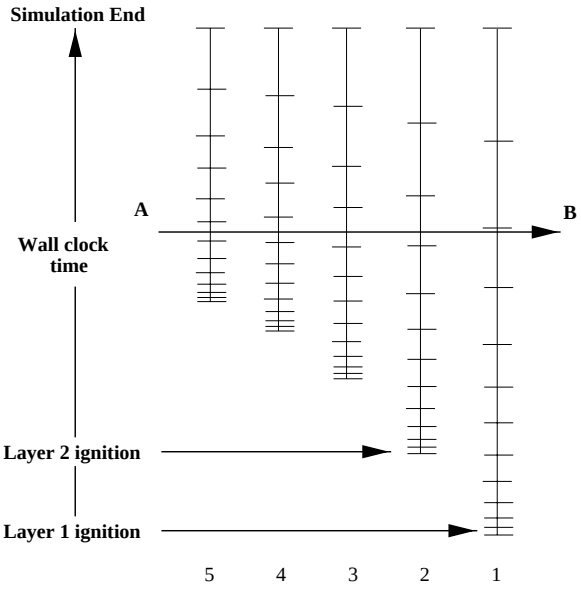

Figure 1.6 An event A in the sixth layer must be carefully tracked through five intervening layers, each with overlapping nonlinear time grids, to propagate correctly to B. The output from layer 6 becomes the input to layer 5 and so on.

ready for its "rightward" neighbor. The Do-loop for layer 4 is started next until its output is brought up to the arrow AB. This process is repeated until finally, layer one uses its Do-loop to process its input up to arrow AB. The output at the end of this last Do-loop becomes the attenuated value associated with the event "A" and is written to the output file. The intricacy of the relay process grows with each newly added layer through this accumulating series of Do loops.

Figure 1.7 further illustrates this cascading Do-loop system in slightly more detail. An event in layer 1 is calculated for each time bin up until the ignition point of layer 2. Calculating layer 1's events is then suspended so that the first event in layer 2 can begin. But now, each event in layer 2 must be processed (attenuated) through layer 1 but not before the calculation in layer 1 is resumed to bring it up to the same wall clock "level" as layer 2. As layer 2 proceeds, each time increment must be coordinated with the time bins of layer 1 until the ignition point of Layer 3 is reached. Layer 3 events must then be coordinated with the previous two layers, layer 4 events with the previous three and so on. In other words, each new layer must not only calculate its own events, but it must ensure each intervening layer is brought up to the same master wall clock level before

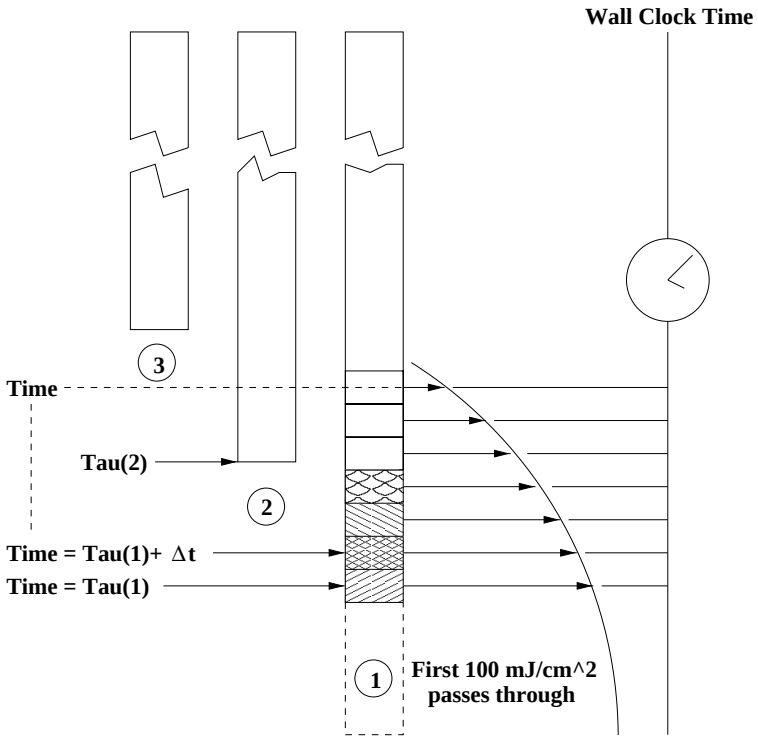

Figure 1.7 After the first 100mJ/cm^2 of fluence, layer 1 activates and processes information directly to output until Tau(2). From Tau(2), the program must process output from layer 2 through layer 1 to output. Layer 3 must process output through layer 2 and layer 1, etc.

calculating its next event. This neighbor–to–neighbor relay, or hand–off, propagates data rightward as the model (by design) incrementally reduces the value of each data point, thereby mimicking the attenuating effects of the induced plasma. The program ends when the wall clock time reaches the end of the pulse.

As the number of layers increases, so does the numerical load. The program uses small time steps, numbering approximately 2,500 for each layer, and, during some parts of the program, the time steps must be rescaled, which further strains the numerical load. Eventually, beyond approximately seven to ten layers, the code begins to yield non–physical results because it is unable to sufficiently resolve any differences in overlapping time scales after a critical number of intervening layers is exceeded.

The old Fortran code is comprised of two sections: one for modeling plasma growth for a single laser pulse, the other for generating intensity profiles by plotting the results from a series of input pulses. This thesis only investigates modifications and enhancements to the single–pulse code. Scattering effects were not included in the old code, which limits it's predictive power. While certainly improving the code's modeling fidelity, the addition would increase the overall computational time since the model is sequential, i.e. depends on a single processor, and thus scattering effects have not been added.

Finally, it should be re–emphasized that this thesis builds on physical insight and the attendant assumptions already laid down by previous researchers after a careful examination of experimental data. Remarks pertaining to these assumptions will be found throughout this work. As was mentioned at the beginning of this chapter, it is not the intent of this thesis to reevaluate the validity of the physical underpinnings of the present model—a task beyond the scope of this thesis; it is to design a more robust computer program which will incorporate ways of extending the limits of the present computer code and so test the assumptions found in the model.

II. Physics

This chapter presents an overview of the essential physics of the carbon suspension cell. It begins with a description of a suspension cell and describes how the code mimics the interaction of the device with an in–coming laser pulse through the use of descrete numerical slices. It then proceeds to a qualitative discussion of scattering and absorption relating to a plasma and bubbles followed with a quantitative development of the essential modeling parameters. Much of the development contained in this chapter follows the development of Kerker [5].

2.1 Essential Parameters

The objective of the code is to model the response behavior of a carbon suspension cell, testing the assumptions underlying the physical response mechanisms. This chapter, however, will not attempt to revalidate these assumptions nor the methods used to derive the simulation parameters but to present them within an experimentally established physical context.

The cell is a thin medium composed of carbon particles suspended in a liquid or solid host, sandwiched between two glass surfaces. The cell is then subjected to a short (10 nanosecond) laser pulse with a power in the range of 10W to 1MW. The code simulates the response characteristics of the carbon black suspension by calculating attenuation cross–sections for two simultaneous, physical processes: 1) a rapidly forming plasma, which contributes to the absorption of the laser energy and 2) relatively slow bubble growth, which causes scattering. The medium of the Primary Zone is divided into descrete cross–sectional slices comprised of a surface area perpendicular to the direction of propagation of the laser beam and a thickness which is a fraction of the thickness of the medium. The cross–section of each slice or layer becomes smaller toward the focus, owing to the geometry of the focusing optics. For any given layer, the original code attempts to cascade its output to neighboring layers lying rightward, toward focus, through the use of an expanding Do loop. Each successive hand-off serves to decrement the layer's original output. The Do loop ends when the rightmost layer writes its results to an output file. Both the old and

new algorithms appeal to the same underlying physical assumptions, which have remained essentially unchanged. These assumptons are referred to as "the model."

Underlying the model's response algorithm is the concept that scattering and absorption from a scattering center is linked to the center's dielectric constant ε_1 and to that of the center's surrounding medium ε_2. The dielectric constant of ethanol is 1.85 and is assumed to be a real quantity. The dielectric constant for a bubble is one and real. A plasma's dielectric constant, however, is complex—a plasma absorbs light. It's absorption is linearly proportional to the imaginary part of its dielectric constant, ε''; scattering is dominated by the real part, ε'. In practice, ε' remains equal to the original host (liquid ethanol) value in cases of interest. Hence, a plasma in this context does not scatter.

In condensed media in the visible, plasma absorption overwhelmingly dominates scattering. There are two limiting cases where this occurs: when the plasma centers are discrete and localized, known as the Rayleigh region, and when they have coalesced into a continuum. The model, as a result, divides absorption from a plasma into a Rayleigh regime and a continuous regime, adding a bridging intermediate regime. All three cases attempt to reflect the changes in the system's response characteristics as a function of size. In this way, the model captures the essential response mechanism of plasma absorption through its functional dependence on ε''.

The code treats bubble scattering by assuming the initial bubble, like the nascent plasma, to be of a size on the order of the carbon particle. The rate equation for bubble growth is, to a first approximation, derived from a large body of research. (See [9], [2], [7], [12] and references therein.) The bubbles will expand adiabatically, i.e. no heat will be added in the initial stages of growth. Small bubbles are limited in their expansion through surface tension, whereas larger ones are constrained through ambient pressure. Integration of the rate equation is performed a priori via Mathcad in a separate computational run resulting in a file of bubble size versus time. The file is then read in by the main program and stored for later use.

As with the plasma, bubble growth is divided into three separate categories based on its initiation and evolution within a liquid: 1) an explosive stage, 2) an adiabatic expansion

stage with an interior of carbon vapor and 3) an isothermal expansion stage with an interior of ethanol vapor. The first stage can be modeled as an underwater explosion of TNT and happens within the first picosecond. This provides the bubble radius and internal pressure as inputs for the second stage. Since bubbles would oscillate without any damping forces, their sizes are kept fixed at their maximum; otherwise, damping is not included in the simulation. The third stage, isothermal expansion, has not been included in the program owing to the emphasis on picosecond time scales.

The code ultimately produces a response curve of intensity versus time, displaying the attenuating effects on the laser pulse as it traverses the sample. Because scattering from bubbles and absorption and scattering due to a plasma are nearly independent, these two processes were uncoupled in the code and computed separately for each time step, their contributions to extinction added together in the exponent of the Beer-Lambert Law

$$I = I_0 e^{-\mu z}, \tag{2.1}$$

where I_0 is the initial intensity propagating over a distance z and μ is the extinction coefficient.

The physics underlying the response mechanisms of the simulation has been unaltered in revising the code; only changes to the computational algorithm were made. Any hard data compiled against the fundamental physical quantities are limited or nonexistent and must be estimated. The methods for arriving at these quantities follow the development in [5] and [11]. The procedure will be to briefly describe these relevant quantities and how they are estimated.

We begin with the cross–sections of the plasma and bubbles alluded to at the beginning of this chapter. Their calculation requires knowing the size of the active volume, the wavelength of the incident radiation and the respective dielectric constants for the active centers and their surrounding medium. The dielectric constant, in turn, depends indirectly on the total energy contained in the scattering medium. The desired input parameters, therefore, are comprised of the energy contained within a scattering volume of a plasma center and the size of the bubble. Complications arise because of the inherently coupled

nature of the competing processes surrounding plasma formation and plasma diffusion. Initially, at carbon vaporization, the plasma is a sea of hot electrons which rapidly diffuse. Diffusion lowers the temperature of the plasma, slowing its growth and changing its absorption cross–section. Together with other losses, this cooling must be offset by laser heating. Bubble growth, on the other hand, is assumed to be much slower and independent of the laser intensity. That is, the light couples *to* the bubble; that is why scattering occurs. The light, however, does not couple *into* the bubble and for this reason, bubble scattering dominates, whereas bubble absorption is negligible. There is an indirect dependency between bubble size and the laser but this is seen at longer time scales. Then, the bubble growth rate increases because the plasma has heated the surrounding liquid. Using the current, approximate bubble growth model of the present simulation, the bubble radius is not coupled to the laser's energy, making possible a more direct and simple calculation of the bubble size. In addition, and as noted above, the dielectric constants for the plasma and bubbles are required. In the case of a bubble, the dielectric constant is one ("free space"). The dielectric constant for a plasma depends on the plasma's density which, in turn, depends on its temperature as illustrated in appendix A.

The quantities most important in the physical description of the cell's response within the present model are then 1) the electron collision frequency and diffusion coefficient, 2) the plasma complex dielectic constant, 3) particle size and 4) particle density.

2.2 Assumptions

The simulation model assumes small, spherical particles with radius a evenly distributed on a simple, cubic lattice. The input is a pulse from a laser with an energy of 1 Joule. The model performs its functions based on the assumption that the carbon particles have been heated to their vaporization point which would require an estimated fluence of 70 mJ/cm^2 This figure can vary, however, with little change to the final results. The program currently uses 100mJ/cm^2 and since the model does not specifically treat carbon heating, the first 100 mJ are assumed to be completely transmitted.

As previously discussed, the model de-couples the calculation of the plasma and bubbles, since their attenuation effects are largely independent of one another. Attenuation

effects are derived from the respective absorption and scattering cross–sections. Initially, a value for the plasma temperature is assumed together with the Saha equation (see Appendix A) to find the density of the plasma electrons. Combining the electron density with the ionization energies of the liquid molecules yields the plasma density which, together with the temperature, determines the complex dielectric constant. Given ε, the cross–sections and finally the attenuation are re–estimated. This interative process continues throughout the pulse.

2.3 Electron Collision Frequency and Diffusion Constant

The electron collision frequency, ν_c, can be estimated from

$$\nu_c = \frac{v}{l_e}, \tag{2.2}$$

where v is the most probable electron velocity and l_e is the electron mean free path. The most probable electron velocity can be found by allowing an expression of the form $F(v)dv$ to represent the mean number of electrons contained in a unit volume with velocity v in the range from v to $v+dv$. Employing the Maxwell velocity distribution, $F(v)$, becomes [8: page 207]

$$F(v) = 4\pi\, n \left(\frac{m}{2\,\pi\, kT}\right)^{\frac{3}{2}} v^2 e^{-mv^2/2kT}, \tag{2.3}$$

where n is the number of molecules per unit volume, m is the mass of the molecule in kg, $k = 1.38 \times 10^{-23} J \cdot K^{-1}$ and $T =$ temperature in degrees kelvin.

The most probable velocity is found from the condition that

$$\frac{dF}{dv} = 0, \tag{2.4}$$

which yields

$$v^2 = \frac{2kT}{m}. \tag{2.5}$$

Hence the most probable velocity is

$$v = \sqrt{\frac{2kT}{m}}. \tag{2.6}$$

The values for $k(\text{erg}\cdot\text{K}^{-1})$, $T(\text{K})$ and $m(\text{gm})$ yield a probable velocity from equation 2.6 whereupon

$$v(cm/s) = 5.93 \times 10^7 \sqrt{E(eV)}. \tag{2.7}$$

where E is now the temperature of the electron in units of electron volts. An electron with a temperature of around 1 or 2 eV will, therefore, have a velocity respectively of 6 or 8×10^7cm/s.

The electron mean free path, l_e, is estimated from the equation

$$l_e = \frac{1}{N q_c}, \tag{2.8}$$

where N is the electron density and q_c is the collision cross–section of the hot electrons with the surrounding liquid molecules. N is found from

$$N = \frac{\rho}{M} N_A, \tag{2.9}$$

where ρ is mass density, M is gram molecular weight of the liquid and N_A is Avagadro's number. The collisional cross–section, q_c, for ethanol and other organic liquids at low pressures have been estimated from data provided by Brown (see [1]). Brown [1: page 13] defines a "probability of collision," P_c, as "the average number of collisions that occur when an electron travels a distance of 1 cm at a pressure of 1 torr at 0°C." P_c depends on velocity in general and has units of an area per unit pressure per unit volume.

A value of approximately 110 for the collisional cross–section of ethanol at 1 eV can be extrapolated from this data (see Brown [1: page 21]), together with the relationship

$$q_c = 2.83 \times 10^{-17} P_c. \tag{2.10}$$

This, along with the value for N, is inserted into equation 2.8 to yield the mean free path of the electron.

The electron diffusion coefficient, D, can be shown to depend on the root mean square of the velocity (see Brown [1: page 99]):

$$D = \frac{1}{3}\langle v \rangle \, l_e. \tag{2.11}$$

The above relationships have been used to estimate the corresponding parameters for ethanol [1]. At 1 eV we have the mean free path of the electron, $l_e = 3.2\text{Å}$, the electron collision frequency, $\nu_c = 1.8 \times 10^{15}/s$ and the diffusion coefficient, $D = 0.71 cm^2/s$.

2.4 Light Scattering

A brief description of the scattering of light from small spherical particles will serve to put into context and emphasize the important remaining quantities of the model, namely the dielectric constant, particle size and particle density.

The model assumes the scattering particle to be a small sphere interacting with a parallel beam of linearly polarized light. It is further assumed the radius of the sphere to be small compared with the beam's wavelength. As a result, the instantaneous electric field inside the sphere is uniform. The field outside the sphere can be described as the superposition of the initial field, without the presence of the sphere, with a field identical to one produced by a simple dipole positioned in parallel to the incident field. The electric field inside the sphere is given as (see [5: pages 31–33]),

$$\mathbf{E}_{int} = \left[\frac{3\,\varepsilon_1}{(\varepsilon_1 + 2\,\varepsilon_2)}\right] \mathbf{E}_0, \tag{2.12}$$

where ε_1 and ε_2 are, respectively, the dielectric constants of the sphere and the surrounding medium and $\mathbf{E}_0$ is the free space electric field. The dipole moment is given as

$$\mathbf{p} = 4\pi\,\varepsilon_2 a^3 \left[\frac{(\varepsilon_1 - \varepsilon_2)}{(\varepsilon_1 + 2\,\varepsilon_2)}\right] \mathbf{E}_0, \tag{2.13}$$

where a is the radius of the scattering center. The polarizability, α', is defined as

$$\alpha' = a^3 \left[\frac{(\varepsilon_1 - \varepsilon_2)}{(\varepsilon_1 + 2\,\varepsilon_2)} \right]. \tag{2.14}$$

The intensity of the scattered wave of wavelength λ and distance r from the scattering particle is indicated by the positional geometry shown in Figure 2.1.

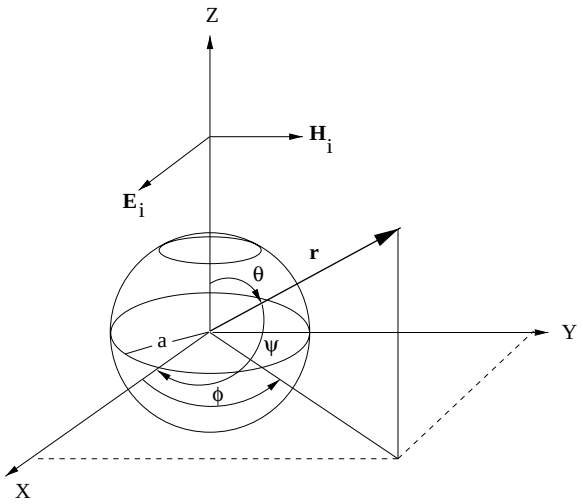

Figure 2.1 Coordinate geometry of the incident electric field in the Z direction and the scattering direction, r, for Rayleigh scattering. The particle with radius a is positioned at the origin. (After Kerker [5: page 33]).

If the incident wave has unit intensity, the scattered wave has an intensity given by (see Stratton [10: page 436])

$$I = \frac{16\pi^4 a^6}{r^2 \lambda^4} \left(\frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + 2\varepsilon_1} \right)^2 sin^2\psi, \tag{2.15}$$

where ψ is the angle between r, the scattering direction, and the dipole on the x axis and λ is the wavelength within the medium. For the present application, the roles of ε_1 and ε_2 in equation 2.15 are reversed from those of equations 2.12, 2.13 and 2.14.

Integrating equation 2.15 over a sphere yields an effective area scattered by the particle called the scattering cross–section,

$$C_{sca} = \int_0^{\pi} \int_0^{2\pi} I r^2 \sin\psi \, d\psi \, d\phi. \tag{2.16}$$

Combining equation 2.16 with equation 2.15 we have [5: page 37]

$$C_{sca} = \frac{128\pi^5 a^6}{3\lambda^4}\left(\frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + 2\varepsilon_1}\right)^2 = \frac{24\pi^3 V^2}{\lambda^4}\left(\frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + 2\varepsilon_1}\right)^2, \tag{2.17}$$

where V is the particle volume ($V = \frac{4}{3}\pi a^3$).

The efficiency of the scattering particle is defined as its cross–section divided by its geometric cross–section, which is $\pi\, a^2$ for a sphere. The scattering efficiency then becomes (see Kerker [5: page 37])

$$Q_{sca} = \frac{128\pi^4 a^4}{3\lambda^4}\left(\frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + 2\varepsilon_1}\right)^2, \tag{2.18}$$

A convenient, dimensionless size parameter, α, is introduced and is defined as the radius of the particle divided by the wavelength of the incident light,

$$\alpha = \frac{2\pi a}{\lambda}, \tag{2.19}$$

which defines the spherical radius using units of $2\pi/\lambda$. Equation 2.18 then reduces to

$$Q_{sca} = \frac{8}{3}\alpha^4\left(\frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + 2\varepsilon_1}\right)^2. \tag{2.20}$$

A complex refractive index describes an absorbing medium in which case it can be shown that the scattering cross–section, q_s, becomes (see Kerker [5])

$$q_s = \frac{8\pi}{3}(n_r k)^4 a^6 |A^2|. \tag{2.21}$$

where k is the wavenumber, n_r is the real part of the refractive index of the medium, a is the radius of the scattering center and A is the polarizability given by

$$A = \frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + 2\varepsilon_1}. \tag{2.22}$$

If the scattering particles are small compared to λ ($\alpha \ll 1$), the absorption cross–section, q_a, can also be shown to be (see Kerker [5])

$$q_a = 4\pi n_r k a^3 I_m[A]. \tag{2.23}$$

where $a^3 A$ is the polarizability of the scattering particle.

Equations 2.21 and 2.23 serve to highlight the strong dependence of scattering cross–sections on particle size and wavelength for the small particle regime. If typical values for the real and complex dielectric constants for ethanol are used, i.e. respectively $\varepsilon' = 2.5$ and $\varepsilon'' = 2.3$, and taking the ratio of equations 2.21 and 2.23 and using the real part of the index of refraction, $n_r = 1.7$, the resulting ratio is

$$q_s/q_a = 0.8(ka)^3. \tag{2.24}$$

Equation 2.24 reveals that absorption will predominate over scattering for $1\mu m$ wavelength radiation for particles with radii less than 180 nm. If absorption is far greater than scattering, then the extinction coefficient, μ, in equation 2.1 becomes Nq where N is the particle density and q is now the total cross–section. The density N can be derived from the inter- particle spacing, l, through

$$N = 1/l^3. \tag{2.25}$$

A typical value for Im[A] as found in the literature is 0.5 and with 1.7 for n, equation 2.23 yields

$$q_a = 70a^3/\lambda. \tag{2.26}$$

If $\lambda = 532$ nm and the particle radius is 15 nm, then

$$q_a = 4.4 \times 10^{-13} cm^2. \tag{2.27}$$

For a transmission T of 70%, a thickness of 1 mm, and using

$$T = e^{-NqL}, \tag{2.28}$$

we would have $N = 8.11 \times 10^{12}$ cm^{-3} and from equation 2.25, $l \approx 0.50\ \mu m$.

For purposes of simulation, the question of whether a plasma primarily absorbs or scatters (expressed through equation 2.24) can be readily answered. At the instant of carbon vaporization, all that exists in the liquid are electrons and bubbles. The scattering cross–section for an electron at low frequencies is the Thomson cross–section which is 0.665×10^{-24} cm^2 (see [4: page 490]); the electron's absorption cross–section is on the order of 2×10^{-17} cm^2 (see [3: page 1459]). Hence, we have the ratio

$$\frac{\sigma_a}{\sigma_s} = 3.0 \times 10^7, \tag{2.29}$$

where σ_a and σ_s are, respectively, the absorption and scattering cross–sections for the electron. The plasma limits through absorption; the bubbles through scattering.

III. Code Description

This chapter focuses on establishing a more complete understanding of the present work by comparing the old and new versions. It examines the set–up of both codes but concentrates on features of the new algorithm designed to overcome the limitations of the old program. The chapter also explores the main computational subroutine, ZAP, which is the same in both versions.

3.1 Previous Code

The old program divides the sample into discrete layers, each layer experiencing carbon varporization at times proportional to its distance from the first layer. The number of layers, N, is only one of a number of input variables the program reads at program start-up. Others include f-number (or f/#), pulse energy, carbon density and particle radius. The input variables are read in from input files, defined as parameters within the main body of the code itself, or defined within an external module which the main program can access. After the input is properly set and the appropriate arrays initialized, the program begins with layer one.

Figure 3.1 is a diagram of the Primary Zone as it pertains to the new algorithm and anticipates a 2D treatment of off–axis scattering which was not addressed in the old version. Both the old and new codes, however, are alike in that both calculate values progressively along the z–axis which is the bottom line in the figure. The final output for both is produced at layer $J = 1$.

Layer one ($J = 1$) is the first to ignite in real time, marking t = 0 for the entire simulation. The old code then begins stepping through two computational loops. The first, an inner loop, calculates the increase in plasma radius and bubble size for a given increment of time for each layer. This yields a value for the attenuation of the intensity using Beer's Law. The second outer loop increments from a given layer to the next layer backward toward the entrance of the sample.

The heart of the calculation for both the old and new versions of the code occurs within the inner loop in a subroutine called ZAP to be described in the next section. The

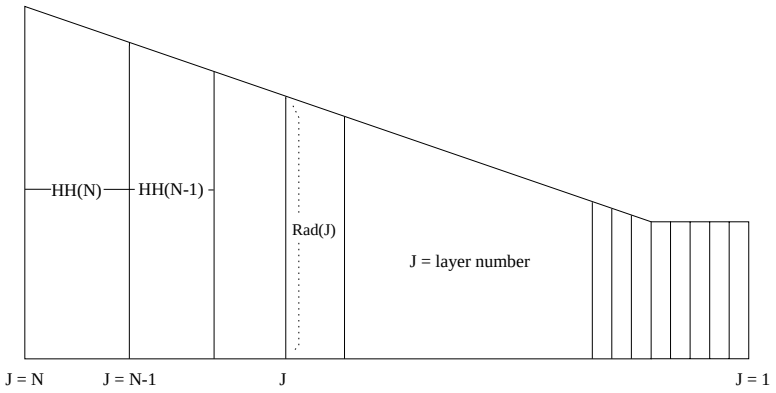

Figure 3.1 Grid geometry of the Primary Zone for the new algorithm. J denotes layer number and HH the inter-layer distance. Rad is the radius for a given layer. The base of the figure denotes the Z axis. For the 1D case, output is calculated at the rightmost point of the Z axis at J = 1.

inner loop of the older version begins when layer J ignites and runs until the end of the pulse. After the first pass of the loop, ZAP is called to create input for the previous layer, layer J-1. The loop then cycles within layer J-1, calling ZAP at each time step. The inner loop moves on to the next intervening layer (previously ignited) but only after residing in layer J-1 the amount of time given by the ignition time for layer J, thus allowing J-1 to attenuate the output from J. Stepping next to J-2, the loop cycles only long enough to attenuate J-1's output, again calling ZAP at each time step, and so on toward the exiting layer, layer 1. (See Figure 3.1.) Since the model assumes no time delay between the ignition time of layer J and its affects on the final exiting layer, the exiting layer's intensity values are paired with layer J's turn-on times and the pairs are written to an output file. The overall effect of cascading the intensity and fluence from J through the intervening layers is to incrementally attenuate J's original intensity and fluence values in proportion to their run of the gauntlet. Once at the exit surface, the inner loop stops and the outer loop steps to layer J+1 and the entire process repeats.

3.2 ZAP

ZAP's purpose is to calculate the extinction coeffcient, μ, for use in equation 2.1. To do this, ZAP's computational scheme is divided into four parts. The first part involves establishing key parameters and defining intermediate variables. Here, the cross–section

for Rayleigh scattering from bubbles is calculated, what the program calls "muiso:"

$$\sigma = \frac{8\pi}{3}\left(\frac{\epsilon - 1}{\epsilon + 1}\right)^2 k^4 r^6. \tag{3.1}$$

where ϵ is the dielectric constant of the medium, k is the wavenumber and r the radius of the bubble.

The remaining parts of ZAP calculate the contribution to the extinction from plasma absorption, what the program calls "muplas." (See Figure 3.2.) The three remaining

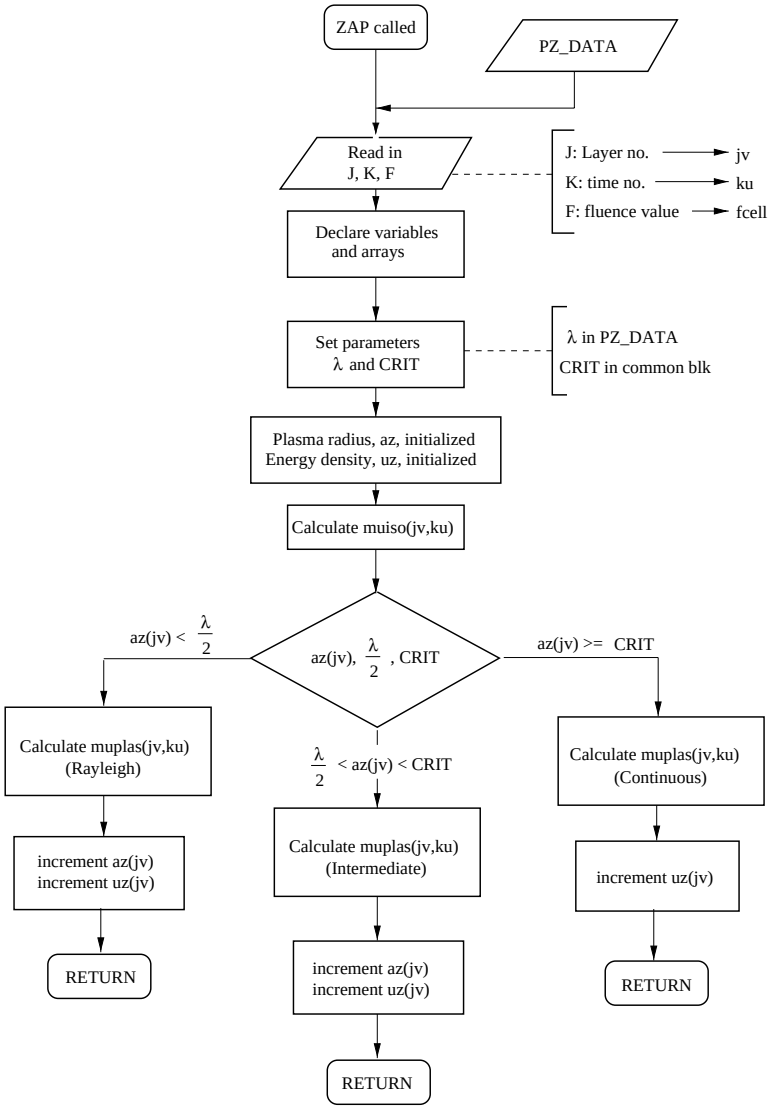

Figure 3.2 Flowchart for the subroutine ZAP.

regions of ZAP are three separate calculational routes. The logic decides which route to take based on comparing the radius of the expanding plasma with the wavelength of the laser. If the radius of the plasma, az, is smaller than $\frac{\lambda}{2}$, use the physics for Rayleigh scattering from small spheres. If the radius has become as large as the separation distance between scattering centers, use the physics for the continuum. The third, intermediate region is a linear extrapolation between the first two.

In all three routes, the process is essentially the same. The scattering and absorption cross–sections are calculated and brought together to obtain μ from

$$\mu = n_p(q_a S + q_{sb}), \tag{3.2}$$

where n_p is the density of carbon particles, q_a is the plasma absorption, S is a stimulated emission factor and q_{sb} is the bubble scattering cross–section. Armed with the bubble radius, ZAP computes the bubble scattering cross–section. ZAP then multiplies this times that fraction of scattered light still remaining within the given layer's collection optics. Next, ZAP finds the absorption cross–section by using the subroutine "sigmaa." The output from sigmaa takes the in–coming fluence, multiplies it by the absorption cross–section, and calls the result "energya." Energya is added to the accumulating plasma center's energy tally to yield the total energy of all the electrons emanating from a single particle. Next, ZAP calculates the diffusion coefficient, "diff". Since diff depends on the temperature, it is updated to the temperature reached at the end of the last time step. Diff is then used to update the plasma radius. (In the continuum region, these last two steps are not calculated since diffusion is not defined for a continuum.) The new plasma radius determines the new plasma volume which is used with the recently updated energy to yield the updated plasma energy density. ZAP then returns to the main program where the energy density, "uz(j)," is used in the SAHA lookup routines "lookupt" and "lookupn" to generate updates for the plasma temperature array, "te(j)", and the density array, "density(j)". (See Appendix A for a more complete description of LOOKUPT and LOOKUPN).

3.3 Present Code

The approach taken to remedy the faults of the previous program, namely the progressive numerical loading and the attendant complex time keeping, resulted in essentially a backward, double-layer hand-off scheme. A single layer takes input from a previous calculation cycle, processes it, and then deposits it into the input array from the next layer. Once accomplished, the present layer's values are forgotten. This constitutes the hand–off, to be explained in more detail shortly. The result is that only two layers need be tracked in time and space during a single computational cycle, allowing the numerical algorithm an increased number of computational layers. The "backward" nature of the scheme derives from the fact that the first layer to be calculated in the code is the last layer to experience plasma ignition, since it is situated at the entrance of the sample.

The advantages of eliminating some complexities inherent in the old code are balanced, however, by new complexities. The new code forces one to think backwards in time. More importantly, it also requires each layer's output to be correctly partitioned from the output of the present layer into the input of the next. (See Figure 3.3). This proves to be rather tricky.

The newer "partitioning" time scheme requires more code to carefully track each layer's progress. This is because each layer now represents a complete, and somewhat independent computational cycle requiring its own set of time and space variables to be properly initialized and tracked. In the older version, each layer's spatial and temporal variables could be accounted for by sequencially incrementing the appropriate loop counting variables through that code's nested Do loop construction. The additional computing overhead of the new code, however, impacts the speed of the simulation. The old version was run on a Zenith 486 when the original code was being written and took approximately two minutes to run. For this project, however, both versions were run on the same IBM machine. Typical timing data for the old and new versions are shown in Table 3.1. The intent of revising the original code was not necessarily toward faster execution but toward eventually parallelizing the code to accomodate the increased demands imposed by adding the rudiments of scattering. Scattering will be simulated along a two-dimensional plane perpendicular to the z axis of the Primary Zone and will require more than one processor.

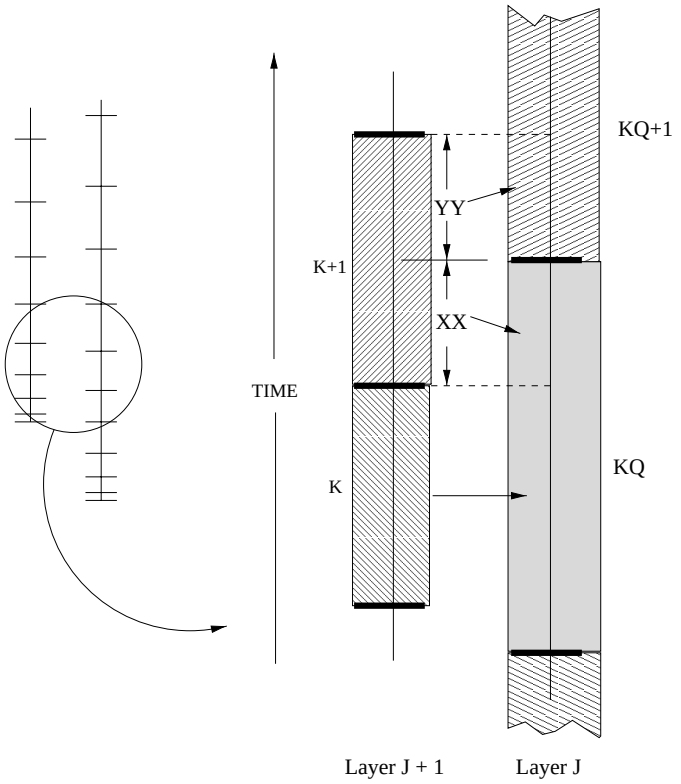

Figure 3.3 The J+1 layer of values is calculated first. Next, layer J starts. But the nonlinear time axis of layer J is offset from that of layer J+1. As a result, some of the K time bins will be contained in a KQ bin and others will fall on a boundary and must be divided. Hence, a portion of the K+1 bin (XX) must be sent to KQ and a portion (YY) to KQ+1.

	OLD	*NEW*
USER	27.74	84.42
SYSTEM	0.05	1.75
TOTAL	27.79sec	86.17sec

Table 3.1 Timing data representative of typical run times for both the old and new versions of the code. "User" is the time spent executing the code. "System" is the time spent performing system services such as accessing the file system, reporting time of day, etc.

The newer code also has quadruple the number of time steps over the original program which has 2,500. This was done to compensate for the progressive loss of data as each layer completed its cycle. Each layer must calculate intensity values for each of its time bins. These values are then assigned to its 'next door' time bins but in a prescribed way. (See Figure 3.3). Because the given layer starts igniting *later* than its previous layer (and therefore offset from its neighbor slightly in time) and because each layer's time bins are partitioned nonlinearly and so get larger with each time increment, a given layer's time bins are smaller than the time bins of the receiving layer. If the receiving layer time bins are much larger than the sending layer time bins, all the sending values become assigned to fewer and fewer receiving bins, in effect crowding the migration of intensity values toward the origin with each new layer. This had the effect of eliminating time bins as time went on and, hence, eliminating data points, yielding intensity plots increasingly sparse toward the end of the timeline and crowded toward the origin. After approximately the 13th layer, the algorithm left all the data condensed along a curve close to the origin and no data after it. The problem was resolved with more and smaller time steps so as to keep the layer–to–layer ratio of time bin size from becoming too large. (For the discussion on the choice of timestep, see Appendix B).

The older version's code rested on a straightforward approach to tracking the progress of the carbon plasma: start at the focal region where layer 1 ignites first and proceed until layer 2 ignites; as layer 2 ignites, incorporate that layer's results into the ongoing calculation of layer 1 until layer 3 ignites whereupon layer 3 is added to the continuing calculation of the previous two and so on.

The newer version, by contrast, starts with the layer that exists farthest from the focal region, i.e. layer 20. (See Figure 3.4.) This is the first layer. The code then projects its output into the next layer closer to focus. This projection or "hand–off", schematically represented in Figure 3.5, is accomplished with each complete pass through the outer main Do loop. At the top of the main loop, the time is reset to begin at the ignition point of the next layer, which becomes the recipient of the hand–off, and the process repeats.

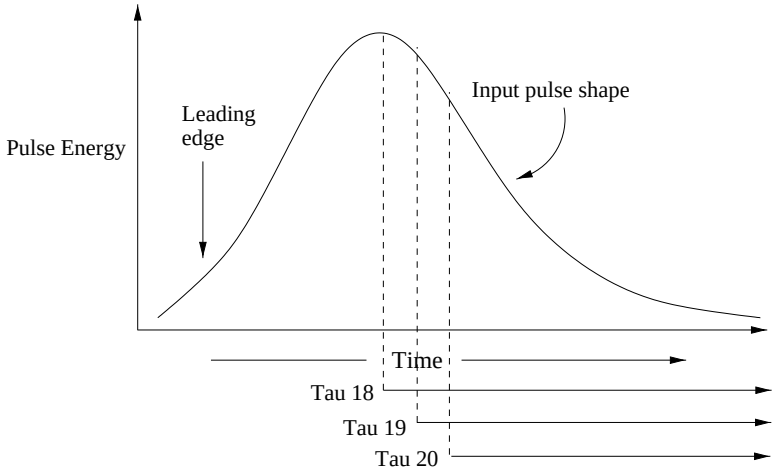

Figure 3.4 This figure illustrates conceptually how the first three layer "histories" overlap and how each successive layer starts earlier in the pulse. The start time for layer 20 is Tau20 and, for its lifetime, processes "free" input only. At the same time the processing for layer 20 loads the input aray for layer 19. The procedure starts over again when layer 19 starts at Tau19. But layer 19 processes "free" input only until Tau20. From Tau20 to completion, layer 19 takes its input from the array loaded by layer 20.

The hand–off between the layers, while obvious, turned out to be critically important to the proper function of the simulation. The algorithm might perform flawlessly within a given layer but yet put its results in the wrong place, destroying the whole process of energy transfer from layer to layer. A fragment of code which performs the first leg of the hand-off for a given layer for one complete cycle of the main Do loop is shown below.

```
Iin(J-1,I)%value = IO(J,I)%value
Iin(J-1,I)%time  = IO(J,I)%time
```

The output intensity variables "IO(J,I)%value" and "IO(J,I)%time" have been calculated for layer J and are being assigned to the corresponding input variables for layer J-1. For the next cycle of the Do loop, layer J-1 becomes layer J, its input variable, "Iin(J,KK)%value", having been modified by MUPLAS and MUISO. That portion of the code is shown next.

```
IO(J,KQ)%value = Iin(J,KK)%value*ARATIO*        &
      DEXP(-MUPLAS(J,K)*HH(J))*DEXP(-MUISO(J,K)*HH(J))
IO(J,KQ)%time = Iin(J,KK)%time
```

This is the second leg of the hand-off: the adjusted value for "Iin(J,KK)%value" and the value of "Iin(J,KK)%time" are assigned to their corresponding output variables which will be read in by the next layer during the next cycle of the Do loop.

If the first layer's output has been calculated (a special case), the second layer starts with another type of calculation until the first layer's ignition time (See Figure 3.6). But since the second layer starts sooner in time, its start time is offset from that of the first. The second layer, (see Figure 3.7), cycles through this offset time receiving unobstructed ("free") laser energy. After the offset, the first layer's output becomes the input to the second layer until the end of the pulse. At the end of the pulse, the input array for the next layer upstream has been loaded and the process repeats.

This chapter has highlighted the main features of the new algorithm: the grid geometry of the Primary Zone, the new timing set–up between two layers, the main computational subroutine, ZAP, and the hand–off routine which allows the results of one layer to be properly transferred to the next. As such, it serves as a motivation for a more complete understanding of the new algorithm and its results, the subject of the next chapter.

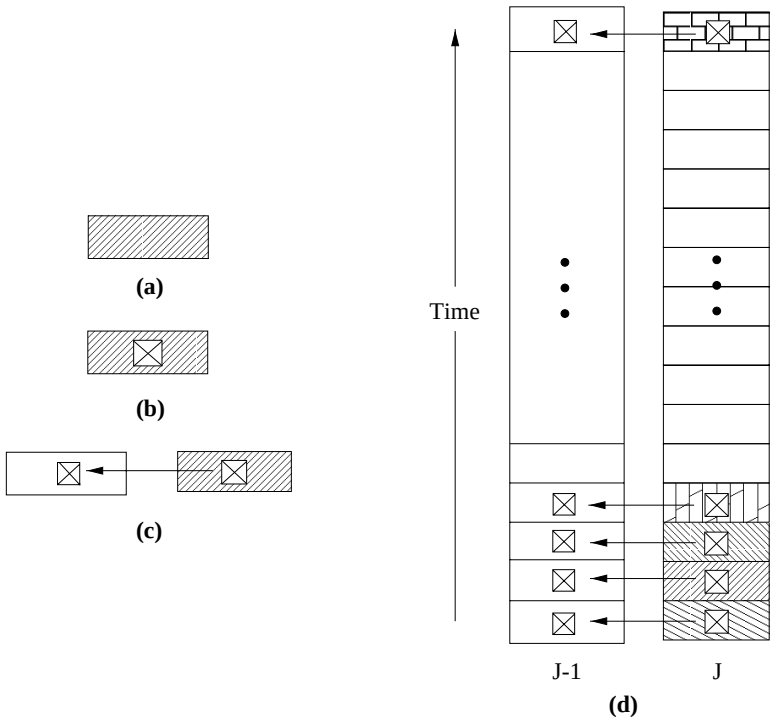

Figure 3.5 Schematic representation of a hand-off. Part (a) represents a conceptual time bin and part (b) shows an output variable calculated for that time bin. Part (c) shows the output variable being assigned to the input time bin on the left. Part (d) displays this pattern of assignment for a series of time bins comprising layer J which is the active layer starting at the bottom and progressing toward the top. The output of layer J awaits layer J-1 when layer J-1 becomes active.

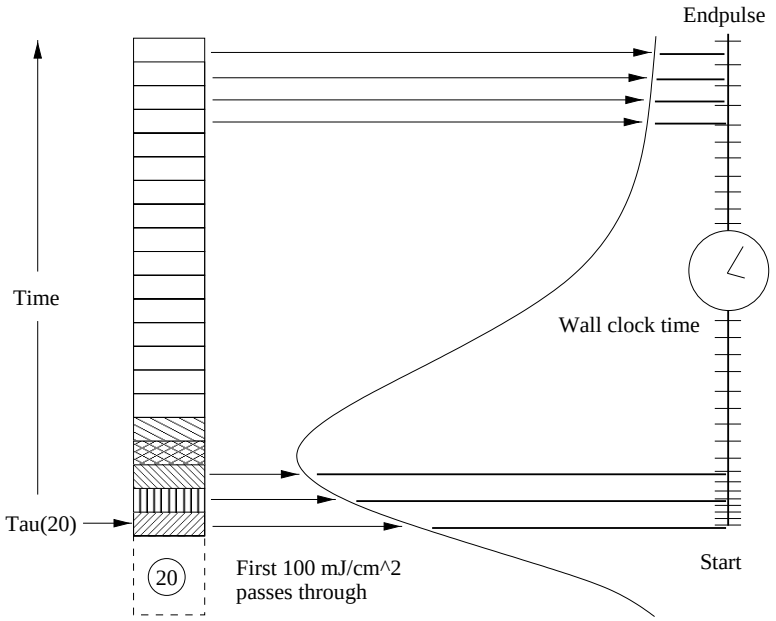

Figure 3.6 The first layer (layer 20) is a special case. It receives only uninterrupted laser energy. After passing the first 100 mJ/cm^2, the layer ignites. The layer then steps through its time bins, loading its output array which yields a plot of the output which is input for the next cycle.

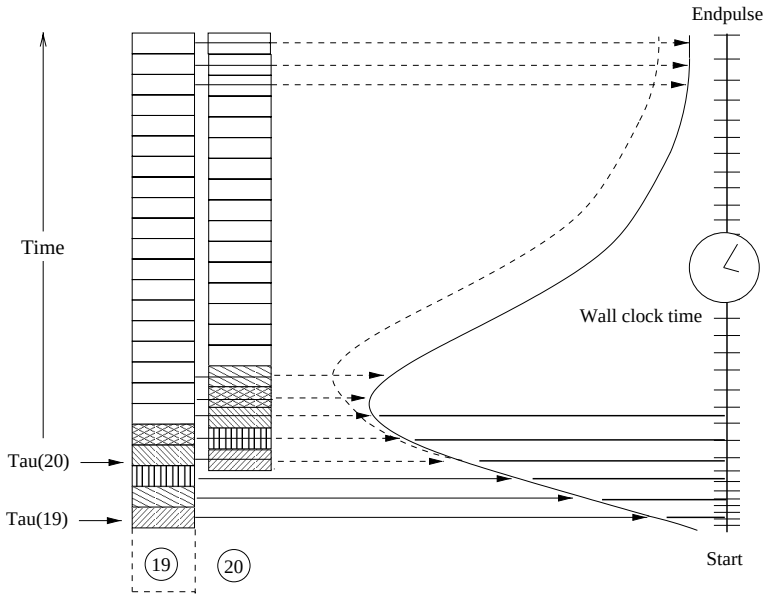

Figure 3.7 Layer 19 starts out receiving free input but soon runs into layer 20. Layer 20's array now becomes the input to layer 19 which loads its arrays. When plotted, they yield an attenuated curve (displayed darker for clarity). Once layer 19 is complete, layer 20 can be ignored for the rest of the pulse.

IV. The Algorithm and Its Results

This chapter discusses the general computational approach used in this thesis. The three special Do–loops—the "A", "B" and "C" loops—are discussed in detail. These form the core of the new algorithm. The chapter concludes with a discussion of the final numerical results which are composed of plots of intensity versus time for a single pulse.

4.1 The Computational Scheme

Except for cosmetic changes, much of the code connected with the subroutine ZAP has been left untouched. This includes the three separate computational routes within ZAP discussed in Chapter 3: the Rayleigh, intermediate and continuous regions. These regions are in the listing in Appendix C. What has changed are the sequence of steps leading up to each call to ZAP. The acts of the play have been rewritten but the main actor, even though he wears a slightly different costume and his cues have been rephrased, still has his old lines.

The general computational scheme can be distilled into a simple flowchart as shown in Figure 4.1. After the program defines its variables and loads the necessary input arrays, it enters a large Do loop controlled by the variable J, the layer number. As seen in Figure 4.1, the loop starts with J equal to 20, which means the program is starting at the entrance surface of the cell. The program then enters a second, nested Do loop called loop "A" which performs the calculations for layer 20 calculations, calling ZAP during each loop cycle. The program uses ZAP to perform similar calculations using the "B" and "C" loops for intermediate values of J and finally writes the results to an output file when J is equal to 1.

Each call to ZAP results in an output which, as discussed in Chapter 3, is immediately employed in the Beer–Lambert equation to calculate the attenuation of the input intensity seen by a given layer over a given time increment. A typical call looks like:

```
CALL ZAP(J,K,F1)
IO(J,I)%value = I1*DEXP(-MUPLAS(J,K)*HH(J))*DEXP(-MUISO(J,K)*HH(J))
```

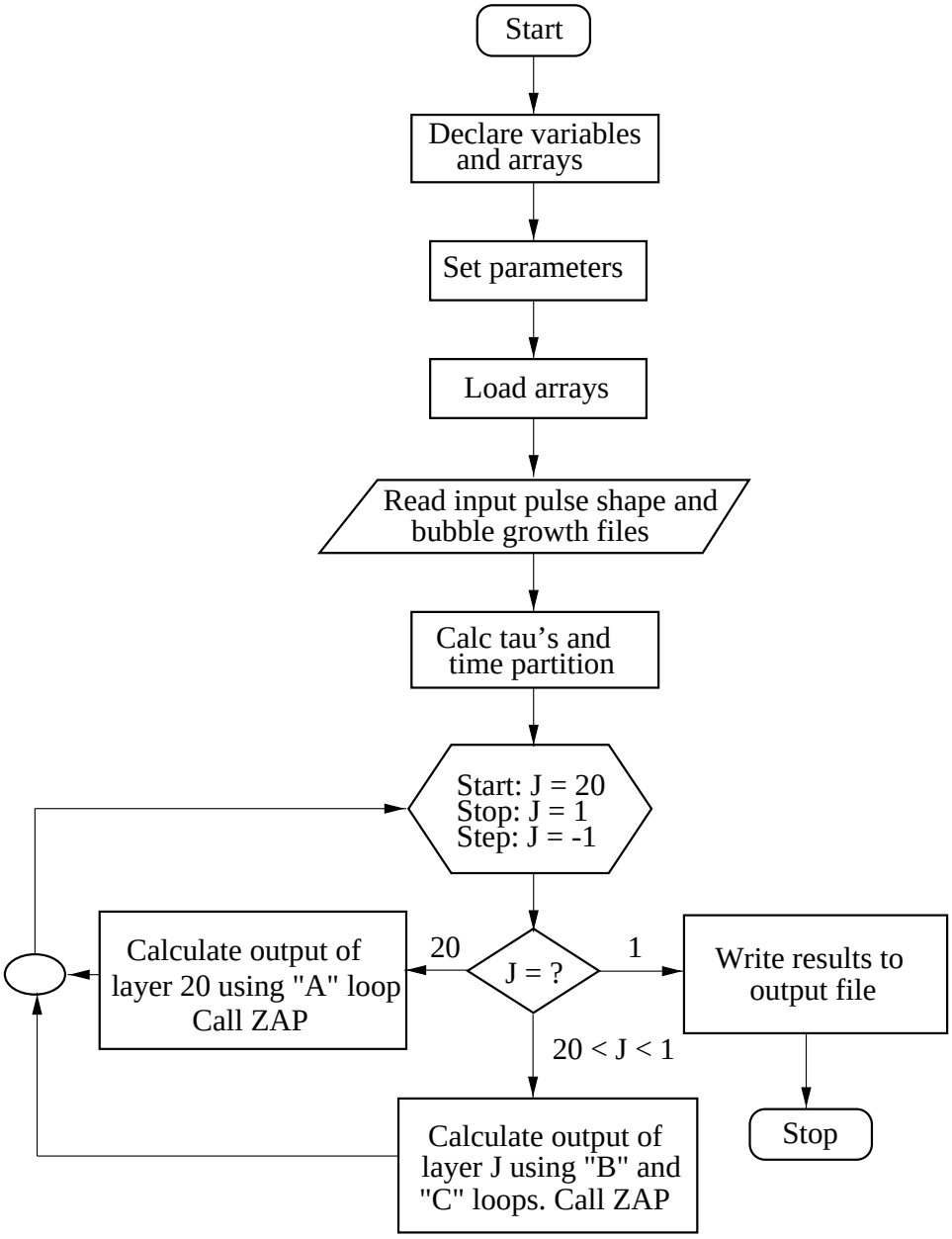

Figure 4.1 Flowchart for the main program.

Here, I1 on the second line is the input intensity being attenuated and IO(J,I)%value is the value of the new output intensity. HH(J) is a thickness of a given layer.

ZAP always receives three arguments: J, K and F1. J, an integer, represents the layer number and therefore is the spatial coordinate. J starts at 20 and ends at 1. K is an integer representing time. K starts at 1 at ignition of the plasma and increments through each time step until the end of the pulse, counting over 10,000 time steps in each layer. K serves as a reference index, coordinating inputs from one layer into the outputs for the next. K also counts into preloaded arrays for bubble growth and plasma expansion. Whatever value K is during the course of the simulation, the program knows the relative sizes of the expanding plasma and the bubble.

The simulation assumes that at the moment of ignition, the spherically–shaped gas of hot electrons and the newly formed bubble have the same dimensions. Both bubble and plasma arrays, therefore, start with the same radius for $K = 1$, which is the first time step. The plasma is assumed to expand faster than the bubble. This means that the hot electrons diffuse outward and hence the plasma radius grows from the center more rapidly than does the bubble radius. As a result, the values of the plasma array, which express a growing radial distance in centimeters, are much larger than the corresponding bubble array values for the same array index counter from the moment of ignition until the end of the pulse. One test of a correctly running program is based on a flag routine, deliberately built into the code, to detect any divergence from these growth rates. The flag routine continuously compares the growth of the plasma and the bubble. Should the K for the bubble array inadvertantly index a value larger than what K indexes in the plasma array (K will be identical for both), the flag becomes set, stopping the program because the bubble can't be larger than the plasma. Finally, F1 is a floating point value representing fluence, where again, fluence is defined as energy per unit area or the time integral of intensity. ZAP, therefore, knows at any moment what layer it is in, where in the history of the pulse it is and how much fluence it has to work with. ZAP takes that bit of fluence, calculates attenuating factors due to the plasma and the bubbles ("muplas" and "muiso" respectively) and, given its time value and layer number, hands these off to the Beer-Lambert Law for attenuation.

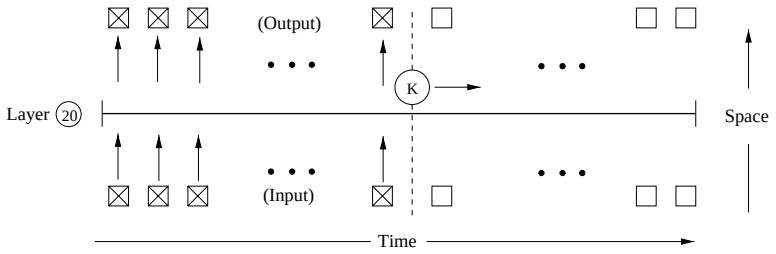

Figure 4.2 Each box in the bottom row represents an input value of direct laser energy. Those boxes filled with an X contain calculated values; those without are waiting to be calculated. As K steps through time from left to right, ZAP attenuates each input and assigns it to an output box on the top to be processed later by layer 19.

The first layer to receive input using the new algorithm is layer 20, the layer at the entrance face of the sample. With no neighboring layer between it and the laser beam to interfere with it, layer 20 receives pure input from the incoming beam (See Figure 4.2). For this reason, the input and output processing of layer 20 is treated differently than layers 19 through layer 2. Layer 1 is likewise processed much like layer 20 since it is the last layer and it too has no neighbor on one of its sides. The processing of input into layer 20 is performed by its own, exclusive Do loop called loop "A". Once loop A is finished, it has loaded the input array for the next layer. Loop A runs only once on layer 20 and thereafter exits the computational scheme. A flavor for how loop A works without going into the details of Fortran can be obtained from looking at the pseudo code for this loop:

var start, end: integer;
procedure loopA
do from start to end
get_next_intensity_value_from_file;
calculate_next_fluence_value;
call ZAP;
calculate_outputintensity;
inputintensity_for_next_layer := outputintensity_of_present_layer;
outputfluence_of_presentlayer := outputintensity_of_presentlayer$\times\Delta t$;
enddo

end

At the top of the loop, a value of intensity is read in from an input file which is then used to calculate the corresponding fluence value. ZAP uses this to calculate in-turn the absorption and scattering cross-sections. These are immediately used to calculate the output intensity which becomes the input intensity for the next layer. Finally, the layer's output fluence is loaded for plotting purposes.

Layer 19 begins a sequence of two loops, performed back–to–back, and repeating from layers 19 down to layer 2. The first, loop B, behaves like loop A except it must coordinate with its follow-on, loop C. Loop B performs its functions over the small interval of time after layer 19 has turned on but before the turn-on time for layer 20. This small offset marks the distinction of the new algorithm from the old: each new layer is offset from its preceeding neighbor in *negative* time—layer 19 starts sooner than layer 20 (See Figure 4.3). Hence loop B works with direct laser input until layer 20 ignites, afterwhich loop C of layer 19 must now work with the input previously created by layer 20. Once loop C is completed, layer 19 has loaded the input array for layer 18 and layer 20 is no longer required. Next, layer 18 sees a small offset from layer 19, whereupon all of the values previously calculated in layer 19 are now inputs for layer 18 and so on.

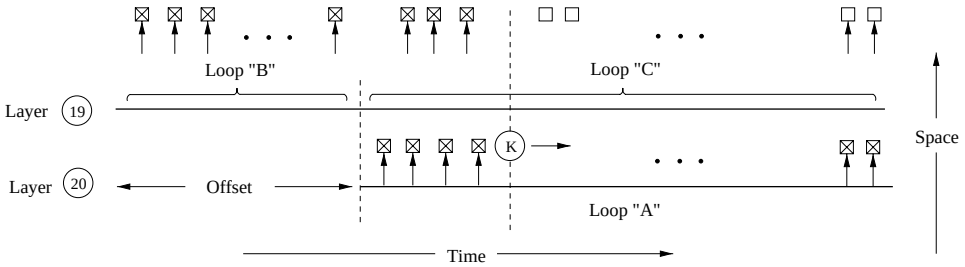

Figure 4.3 After the offset covered by loop B, loop C must now process input previously loaded by loop A.

Loop B is similar to A with the exception that it must "pivot" on its last values and coordinate these with the beginning of loop C, which is taking its input from a previous layer:

var start, loopBend: integer;

```
procedure loop B
        do from start to loopBend
            get_next_intensity_value_from_file;
            calculate_next_fluence_value;
            call ZAP;
            calculate_outputintensity;
            inputintensity_for_nextlayer := outputintensity_of_presentlayer;
            outputfluence_of_presentlayer := outputintensity_of_presentlayer×Δt;
            if loopvariable = loopBend then
               partition_timebin
               loopB_gets_XX_share
               process_with_ZAP
               exitloopB
            end
        enddo
end
```

When loop B is at the end, part of the input fluence into the present layer will come directly from the laser, XX, and the other will come from the output from the previous layer, YY, (See Figure 3.3). Once the XX portion is processed by ZAP, loop B hands its function over to loop C which starts by processing its YY portion.

```
var loopCstart, loopCend: integer;
procedure loop C
        processYYfromloopB
        do from loopCstart to loopCend
            get_next_fluence_from_last_layer;
            adjust_next_fluence;
            call ZAP;
            outputintensity := inputintensity_from_last_layer×attenuation;
            inputintensity_for_next_layer := outputintensity_of_presentlayer;
```

outputfluence_of_presentlayer := outputintensity_of_presentlayer$\times\Delta t$;

enddo

end

Loop C is the primary computational loop for the new algorithm. During each cycle, it must receive input from the previous calculations and partition them into the proper time bins of the next layer. With each successive hand-off, the output becomes more and more attenuated until finally, the output of layer 1 displays the properly attenuated curve.

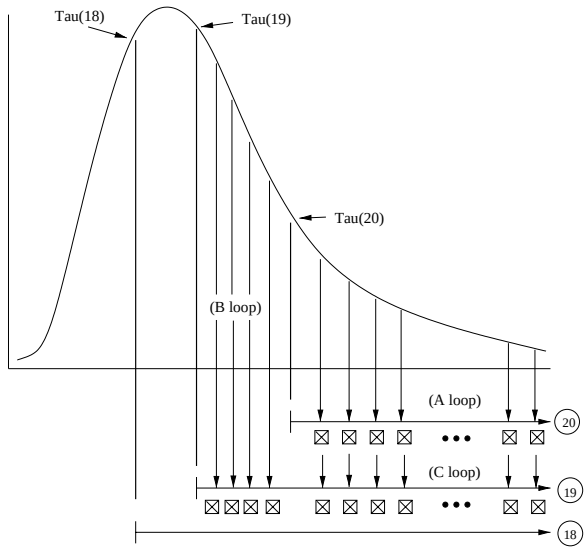

Figure 4.4 Conceptual positions of the A, B and C loops as they pertain to layers 19 and 20. Once layer 20 is computed, layer 19 and the rest of the layers only experience loops B and C.

All three loops are shown as they might appear related conceptually to the profile of a single pulse for the first two computational layers, as shown in Figure 4.4. Layer 20 has the least amount of pulse to deal with, since it is the last to start in "real" time. Layer 20's A loop processes its input and then loads its output array, which is shown in the figure as X'd boxes below the arrow for Layer 20. Next, layer 19 has loop B load a portion of its array directly from the pulse. Then, shortly thereafter, at tau(20), the C loop takes over and processes the output boxes from layer 20, turning them into output boxes below the arrow for layer 19. This double layer arrangement repeats itself, with each successive

cycle starting closer to the origin. The final layer's output simply reports the accumulated results of the preceeding ones to an output file.

4.2 Results

With both the old and the new codes receiving the same input pulse shapes (see Figure 4.5) and the same set of input parameters (see Table 4.1), both exhibit nearly the same output shape—a sharp rise in the output pulse with a precipitous cutoff early in the pulse, albeit with a slightly greater peak intensity in the old version. (See Figures 4.6, 4.7 and 4.8).

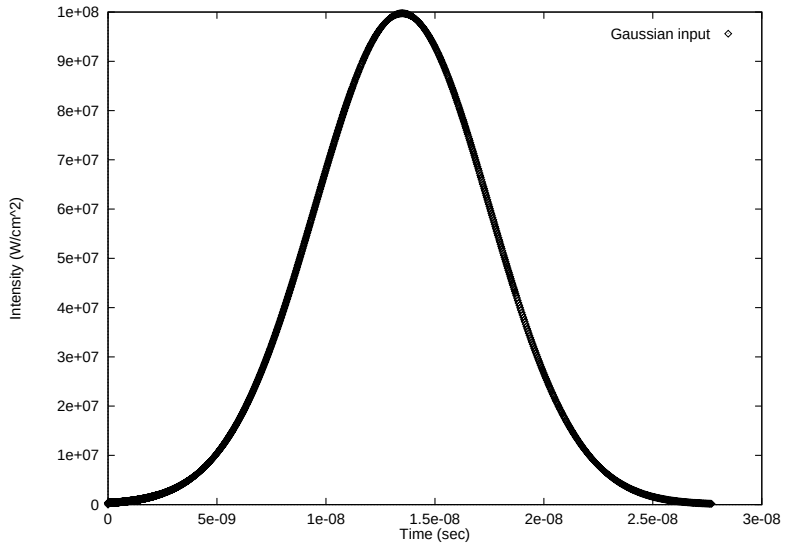

Figure 4.5 The input to the simulation is gaussian instead of a Q-switched pulse shape due to an external subroutine which uses the readily available formula for the Gaussian distribution.

An explanation for the slight differences seen in Figure 4.8 is possible. Apart from the algorithms of the two codes, the only difference between the two versions is the number of time steps—2,500 in the old versus 10,000 in the new. As a result, events in the new code are being captured in time four times more accurately. This would reasonably have the effect of much more precisely locating an event in a time bin for a given layer with the corresponding time bin of its receiving layer. Consequently, each layer's output would progress within much more narrowly aligned time limits from beginning to end. With no

Name	*Symbol(dimensions)*	*Value*
Particle Radius	a_0(cm)	8.00×10^{-6}
Gamma	γ	1.33
F-number	f/#	5
Number of Particles	$n_p(/cm^3)$	1.0×10^{10}
Diffusion constant	dconst(cm^2/s)	0.7
Laser Pulse Width	width(nsec)	9.0
Liquid Ionization Potential	Chi(1)(eV)	12.2
Liquid Ionization Potential	Chi(2)(eV)	14.0
Liquid Ionization Potential	Chi(3)(eV)	16.0
Laser Pulse Energy	Epulse(Joules)	6.0×10^{-5}

Table 4.1 The input data set used for both the old and new codes. Gamma is the ratio of the heat capacity at constant volume to the heat capacity at constant pressure. F-number is the ratio of the focal length of a lens to its diameter.

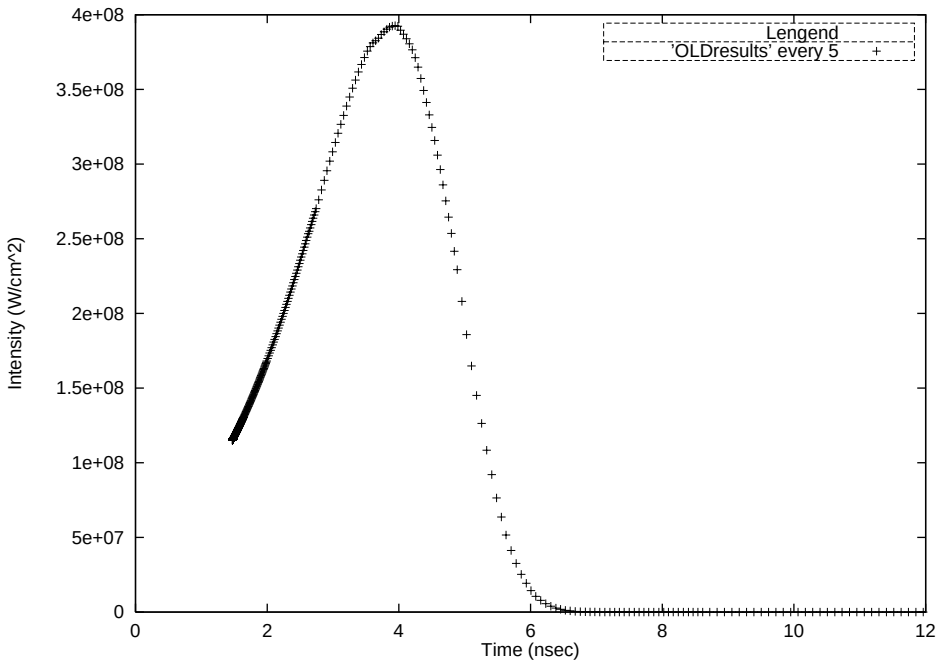

Figure 4.6 The output of the older version of the simulation. Note: in each plot profile, the timeline for the in–coming pulse extends to 28 nsecs.

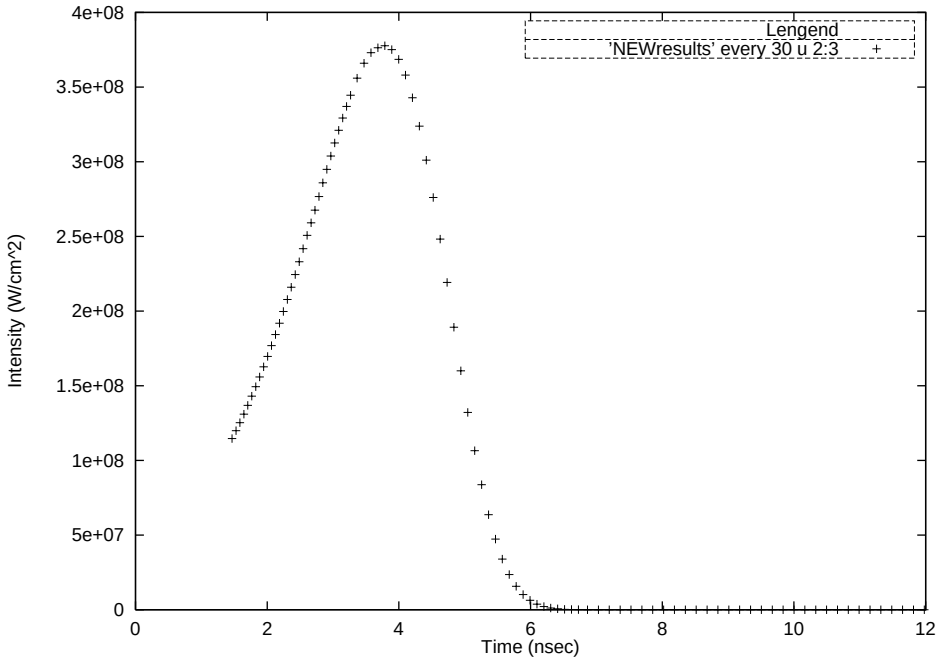

Figure 4.7 The output of the newer version of the simulation.

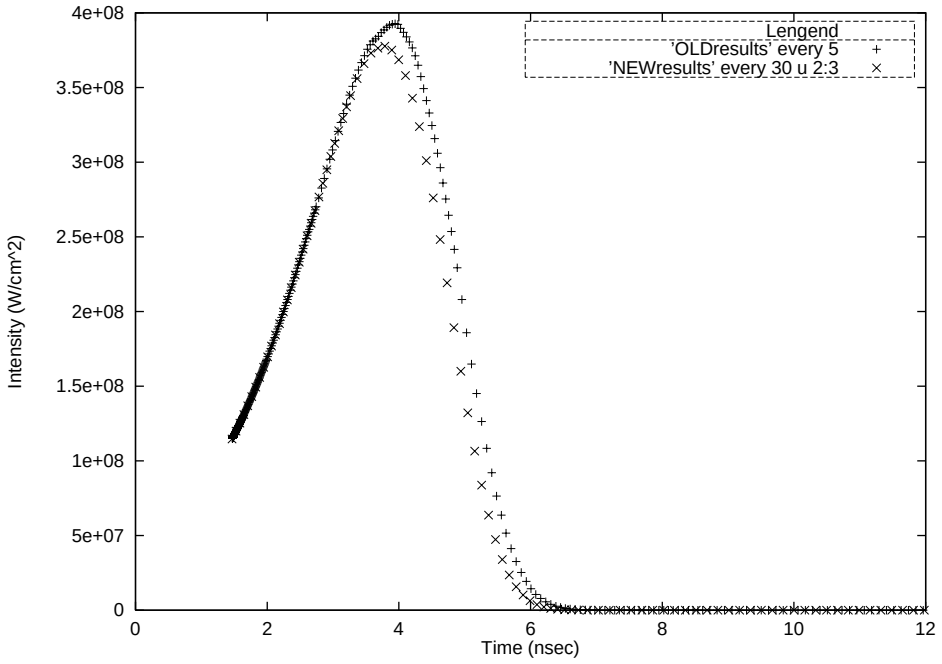

Figure 4.8 Both the old and new outputs are plotted together. Note the close similarity of both plots.

'gauntlet' to run as in Figure 1.6, the effect would be to shift the final output closer to the actual time values compared with the more coarse time divisions of the old code and with a correspondingly greater accumulated attenuation.

Figure 4.8 and those following demonstrate that the simulation of each code is based on the same model and that the algorithms of each are equivalent. Both reflect the same trend when undergoing changes to the input pulse energy (See Figures 4.9 and 4.10). A plot of output intensity from the old code as a function of plasma radius is shown in Figure 4.11 and for the new code is shown in Figure 4.12 . The response effects of the new simulation develop consistently from the front to the exiting surface where Figure 4.13 displays the effects of the simulation on selected intervening layers between the entrance (layer 20) and exit (layer 1) surfaces.

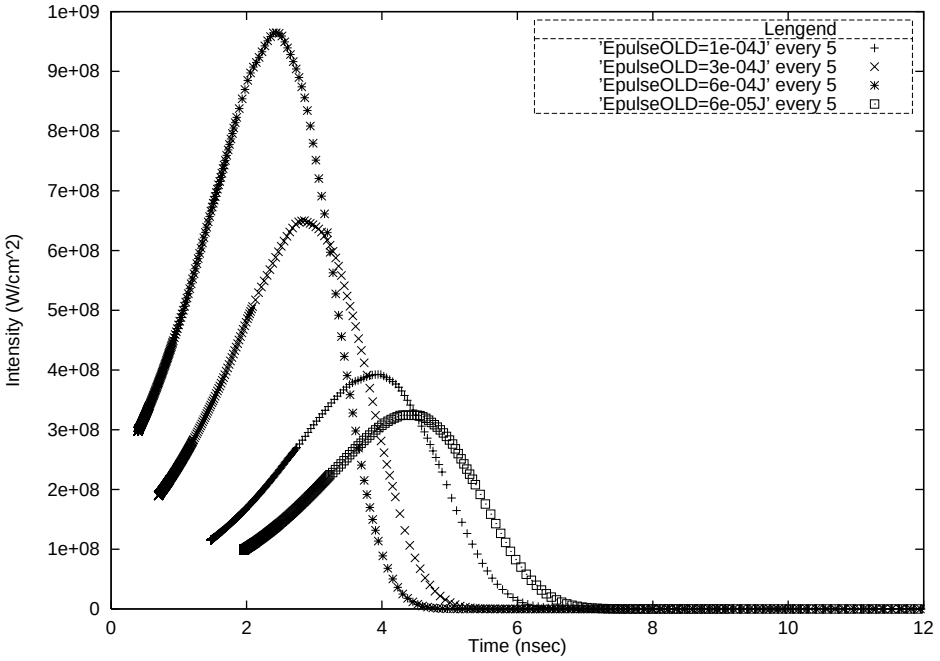

Figure 4.9 Output intensity as a function of pulse energy for the older version.

The new algorithm is by no means perfect. It is slower than the old version, but not significantly slower. The new algorithm requires each layer to be treated separately which necessitated that the RBUBR array loop be filled by interpolating between the elements of the RBUB array *within* the MAIN loop. Now, each new layer number J requires a new

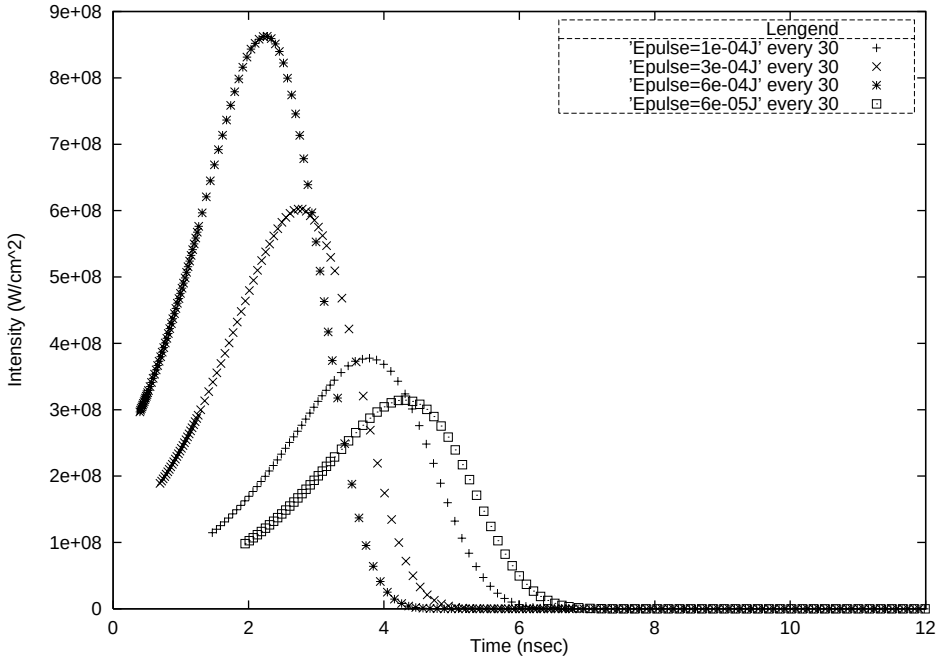

Figure 4.10 The output intensity of the new version as a function of pulse energy.

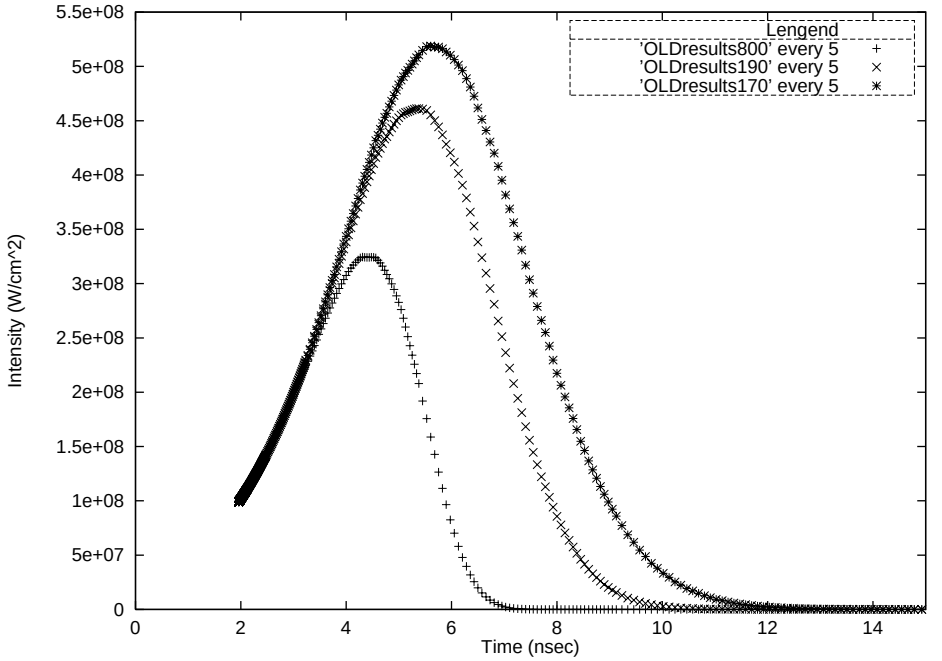

Figure 4.11 Old output as a function of plasma radius, A0. The bottom curve is for A0 = 8.0E-06 cm and is included as a baseline for comparison with the curves above it generated from the smaller values of A0. In the legend, 800 is 8.0E-06 cm, 190 is 1.90E-06 cm and 170 is 1.70E-06 cm.

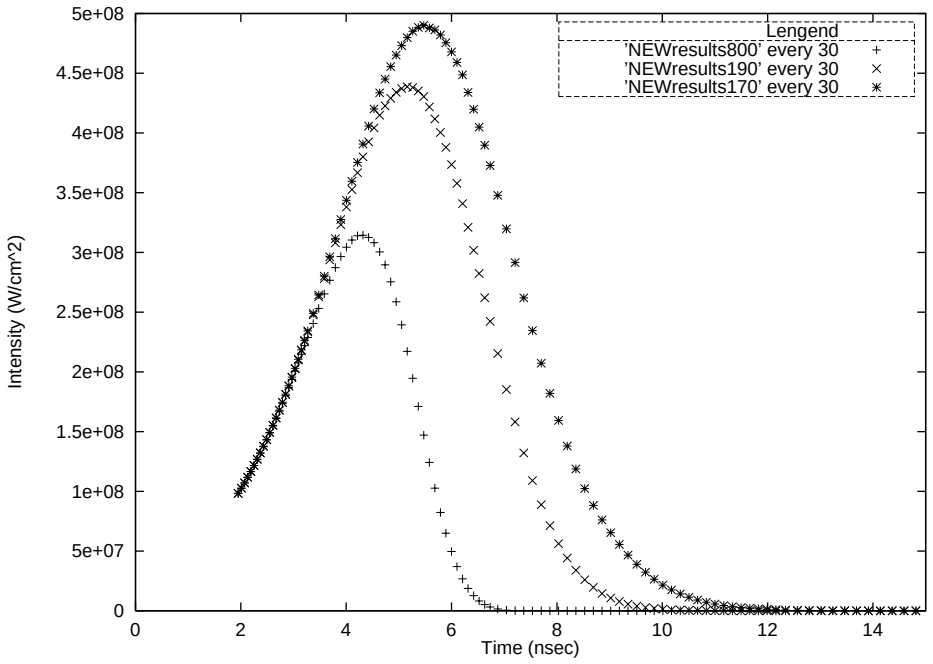

Figure 4.12 New output as a function of plasma radius, A0. The bottom curve is again for A0 = 8.0E-06 cm with the same legend designation as in Figure 4.11.

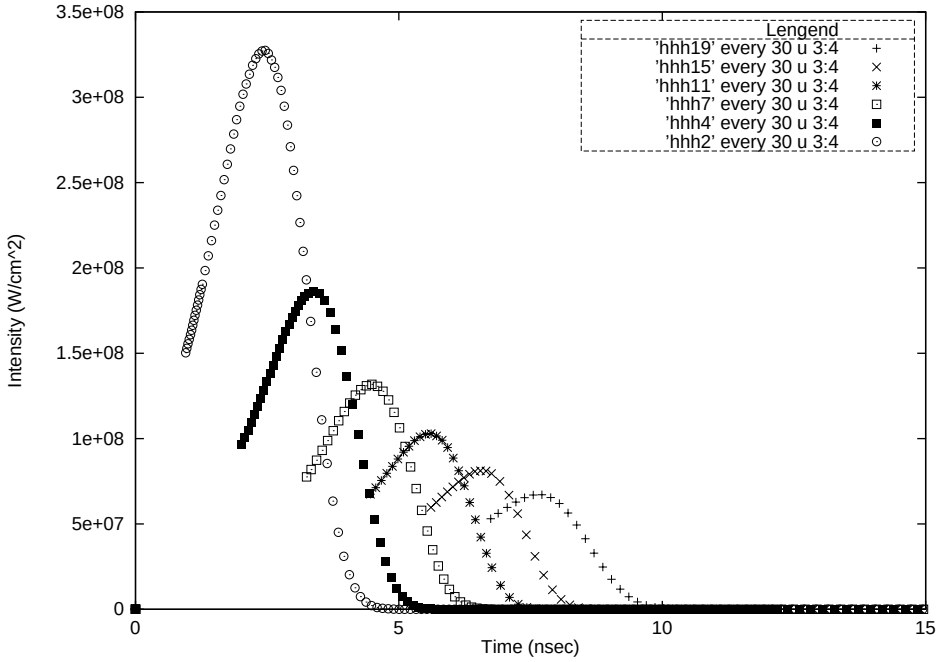

Figure 4.13 The effects of selected layers on a single pulse are shown superimposed for layers 19 through layer 2. In the legend, xy in hhhxy refers to the layer number. The curves closer to the origin are layers nearer to focus.

RBUBR array to be filled, whereas in the old code, RBUBR is filled only once. Together with quadruple the number of time divisions, the new program is bound to run slower. To create a faster code in the future, arrays keyed to each layer will have to be filled first outside of MAIN. Then each layer-specific bubble array can be called directly from within MAIN. The intent of this thesis, however, was not necessarily toward faster execution (although that certainly would have been a plus) but towards creating a code that could be extended to two dimensions and could accomodate parallel programming techniques while still retaining the essential features of the original code for testing purposes.

The algorithm still suffers from a number of limitations. For example, small decrements in the plasma radius, a0, near 1.7E-06 cm produce anomalous results due, in part, to an incomplete development of the bubble growth files which support this size regime. The equations for bubble growth which have been incorporated into the code have not been well developed and will therefore need to be revised. In addition, adiabatic growth for bubbles in short time scales displays a strong dependence on the ratio of specific heats of the surrounding liquid and the vapor of the interior, namely γ. This functional relationship has not been written into the program and will need to be addressed in the future. The code also suffers from an inability to properly accomodate the effects of scattering, which when accomplished within a parallel construction, should account for much more light re-entering the Primary Zone, further modifying the output intensity profile. Finally, the liquid will boil at high energy densities, which is another feature that needs to be addressed.

Overall, however, the aim of this thesis project has been largely successful. Errors have been uncovered and corrected. For example, values of the plasma radius were not being properly accumulated, and so the formula for the plasma radius in ZAP had to be adjusted. It was also discovered that the old and new codes were not being compared using exactly the same starting conditions. This involved the values of ARATIO for the old and the new code where ARATIO is defined as the ratio of the area of the $J+1$th layer to the Jth layer. It proportionally increases the intensity seen by the Jth layer as the simulation moves to layers closer to the focal region. While the values for ARATIO had each been calculated correctly, the timing of their use was offset by one layer between

the two versions of the code. This too was corrected. Refinements have been built into the code in the form of Fortran 90 features. Most importantly, the vexing limitation inherent in the old code, that of an upper limit to the number of layers a sample could be divided into, has been largely overcome. This allows a more reasonable confidence level to be attained for $N > 10$, where $N =$ the layer number. This is important for the larger task ahead, that of scaling the simulation to two dimensions. It is then that the more important aspect, largely ignored in the present effort, can be seriously addressed, namely scattering.

V. Conclusion

A new approach to simulating the response behavior of a carbon black suspension cell has been developed employing a double–layer hand–off algorithm. The response mechanism results from the coupling effects on a laser pulse with a rapidly expanding plasma. The plasma is generated from the ignition of carbon aggregates suspended within the liquid sample while being irradiated by the incoming pulse. As the plasma expands, the pulse becomes increasingly attenuated such that points beyond the exit surface of the sample are shielded from the effects of the pulse by virtue of the interaction between the plasma and the laser.

The new approach is based on physical insight forged from earlier experimental work. It is outside the scope of this thesis to re–examine the physical basis of the code and its assumptions—what has been termed "the model". The intent of this thesis was to modify and extend the numerical algorithm 'surrounding' the model in such a way that it's assumptions might be tested through the future developement of a 3–D code using concurrent programming techniques.

Like its predecessor, the new scheme divides a sample into discrete layers, transforming inputs into slightly attenuated output. The old version of the code suffered from a cumbersome tracking mechanism which eventually constrained the numerical computation. Unlike its predecessor, however, the new algorithm chooses the first layer as being the farthest in space from the site of plasma ignition (the focal region) and the last in time to ignite. Each layer is therefore set in motion with unique time and space variables which create its own local 'history'. The algorithm superimposes each layer's history in such a way that once a given layer's output has been properly prepared and processed, tracking that layer is no longer necessary, allowing a greater number of layers to subdivide a sample thickness. Whereas the old version typically allowed from seven to ten layers, the new version routinely runs twenty.

The new algorithm suffers from a number of limitations. The algorithm is not faster than the old version. This is because, with everything else equal in terms of initial parameters and data sets, the new code has four times the number of time steps with four times

the number of calls to the subroutine ZAP. This was done to eliminate the data 'drop out' seen in the initial output plots when the new algorithm was being developed. The code, therefore, will have to be modified to run more efficiently. For example, one way will be to explore numbers of time steps less than 10,000 which still prevent data drop out yet maintain the same time domain structure for capturing the essential temporal dynamics. Another option is rather than customizing the RBUBR array within MAIN's Do loop for each J, RBUBR will be loaded for each J beforehand. This will require a two-dimensional RBUBR array. Also, the number of time steps will have to be reduced which means addressing the issue of the cancelling time bins. The code also suffers from a lack of a refined treatment of bubble growth and boiling. But perhaps the most significant flaw in the present code is a lack of a description for scattering which afflicted the old code as well. Largely ignored due to time constraints, scattering will need to be seriously addressed in the future if the new code is to be realistically predictive. The reliability of the present code is addressed through Table 5.1 which lists the ranges of input variables within which the simulation, in its present state, can be expected to produce physically valid results.

Name (symbol)	*Range*
Particle Radius (a_0)	10nm to 100nm
F-number (f/#)	1 to 50
Particle Density (n_p)	10^{10} to $10^{14}/cm^3$
Diffusion Const (dconst)	0.3 to 7 cm^2/s
Pulse Width (width)	1ns to 30ns
Ionization Potential (χ)	7 to 14 eV
Plasma Temperature (TE)	0.5 to 20 eV
Intensity (Iin)	$\leq$10GW/cm^2

Table 5.1 The range of input variables determining the validity of the present code.

Scattering effects will require a two–dimensional treatment. The plasma is composed of electrons. The bubbles, however, cannot be ignored because they are located at the center of each cell. They become scattering centers, growing larger throughout the lifetime of the plasma. They cause a significant amount of light to be redirected back into the Primary Zone. Presently, the Primary Zone "exists" only as the z-axis. To mimic the action of redirected light, scatterers off the z-axis will be required, which means a two-

dimensional array of points will be set up to span the roughly cone-shaped Primary Zone. (See Figure 5.1).

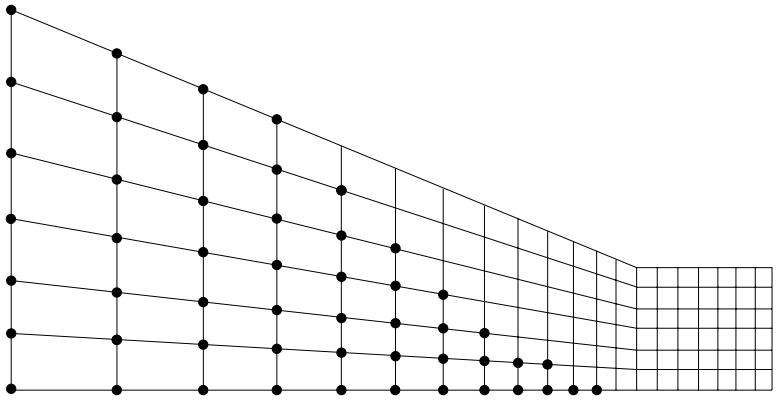

Figure 5.1 Off-axis scattering will be accomplished through an array of node points roughly spanning a cross-section of the Primary Zone. Only the upper half need be used because of symmetry. The bottom line is the present z-axis.

With the array in place, a scattering transfer function will be imposed to approximate light scattering into the Primary Zone. The initial transfer function will impose a 3–1–3 arrangement of input to output, as shown in Figure 5.2. The top and bottom boundary lines of the Primary Zone will have a 2–1–2 scheme. Each array point will be designated a position relative to the origin of the grid. The three leftward neighbors will supply input and the three rightward neighbors will receive the output in proportion to their respective positions on either side of the (i,j)th grid point, as shown in Figure 5.2. The circular symmetry of the Primary Zone will allow the array of grid points to simulate the entire volume as shown in Figure 5.3.

The present work is a start. The code will become truly useful only as far as it faithfully predicts well known material response behavior. The code will have attained such a predictive capability and with reasonable confidence if, given a set of well–defined starting conditions, it can simulate the observed phenomena. Once this 'retroprediction' goal is attained, the code can become a useful "what if" tool for the materials scientist or engineer. For example, researchers inevitably want to know which boundary conditions of a device design favor one response over another. Specifically, they are interested in knowing the kind of behavior a material would exhibit if the particles of the suspension

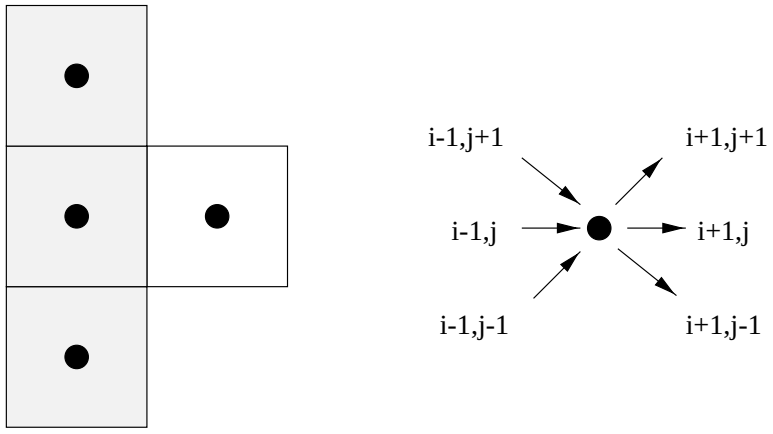

Figure 5.2 Part (a) displays three shaded grid points which supply input to the single unshaded neighbor grid point. The symmetric input/output scheme is shown in terms of grid position in part (b).

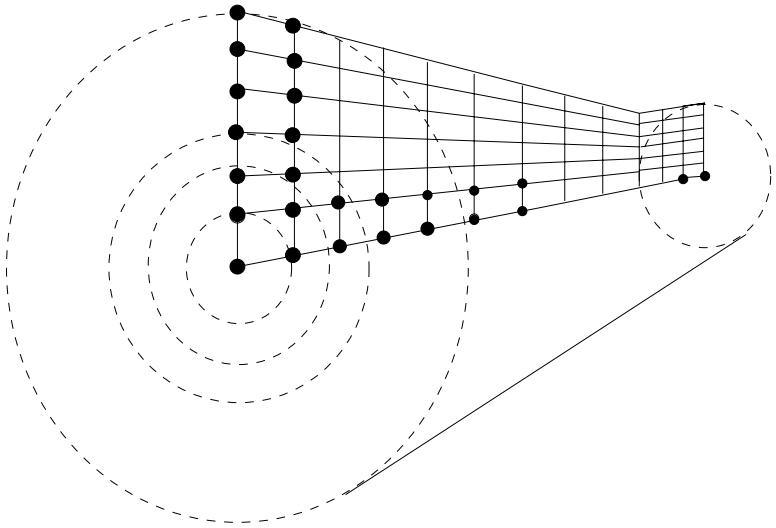

Figure 5.3 The cross-sectional plane of grid points, when rotated around the z-axis, will span the roughly cone-shaped Primary Zone volume.

cell were smaller. The present code has already shown it has the potential to answer that question in a semi-qualitative way, thereby showing promise as a guide in helping materials researchers in their on-going experiments.

Appendix A. The Saha Equation

The Saha equation governs the thermal ionization of carbon particles within a volume of a plasma under conditions of local thermodynamic equilibrium and is an extension to the Boltzman relation. It is this thermal ionization of carbon atoms which forms the basis for the material response under investigation in this thesis. The Boltzman distribution will be discussed first. It describes the fraction of atoms having a certain quantum level within the volume of a plasma. The Boltzman formula will then be extended to include consecutive ionization levels of an atom, which leads to the Saha relation. The means by which the code actually implements the SAHA equation and a listing of the code itself completes the appendix. The development is taken from Lochte [6].

If a volume of a plasma contains the number n of identical particles in a cubic centimeter, then some fraction will exist in an excited state. If that fraction is designated n_i and they occupy the i^{th} quantum level having energy E_i and assuming thermodynamic equilibrium, then n_i can be described by the Boltzman equation,

$$\frac{n_i}{n} = \frac{g_i}{U(T)} \exp\left(-\frac{E_i}{kT}\right), \tag{A.1}$$

where g_i is a statistical weight for the i^{th} level and $U(T)$ is the partition function,

$$U(T) = \sum_i g_i \exp\left(-\frac{E_i}{kT}\right). \tag{A.2}$$

The summation must include all levels including the highest existing level. This last energy level is given by

$$E_j = \chi - \Delta\chi, \tag{A.3}$$

where χ is the ionization energy.

We now look at ionized atoms and consider the numbers n_z and n_{z-1}, which represent the number of particles occupying respectively the z and $z-1$ ionization levels of an atom. z is the number of electric charges seen by the radiating electron and $z-1$ is the charge on the ion or atom. By comparing two consecutive ionization levels we can extend the Boltzman relationship to these ionized atoms. The ratio of the number of atoms in the

$z-1$ level to those in the z level is given by the SAHA equation,

$$n_e \frac{n_z}{n_{z-1}} \equiv S_{z-1}(T) = 2\,\frac{U_z(T)}{U_{z-1}(T)}\,\frac{(2\pi\, m_0 kT)^{\frac{3}{2}}}{h^3}\exp\left(-\frac{\chi_{z-1}-\Delta\chi_{z-1}}{kT}\right), \qquad \text{(A.4)}$$

where n_z and n_{z-1} are particle densities and n_e is the density of free electrons per cm^3. The two partition functions, U_z and U_{z-1}, correspond to the two ionization levels and m_0 is the electron's rest mass. If the ionization energy is given in eV and the temperature in degrees Kelvin, equation A.4 reduces somewhat to a more practical expression,

$$\frac{n_e n_z}{n_{z-1}} = \frac{U_z(T)}{U_{z-1}(T)}\,4.83\times 10^{15} T^{\frac{3}{2}}\exp\left(-1.1605\times 10^4\frac{\chi_{z-1}-\Delta\chi_{z-1}}{T}\right). \qquad \text{(A.5)}$$

The ratio of the partition functions in equation A.5 reduces to 1/2. The code also supplies a series of constant ionization potentials, χ_1, χ_2 and χ_3, and treats the $\Delta\chi$s as a constant, yielding an effective χ. The value of $\Delta\chi$ is estimated at between 1 and 1.5 eV. Simplifying the exponential term, we have

$$\frac{n_e n_z}{n_{z-1}} = 2.4\times 10^{15} T^{\frac{3}{2}}\exp\left(-\frac{\chi_{z-1}}{(8.617\times 10^{-5})T}\right). \qquad \text{(A.6)}$$

Solving the SAHA equation for temperature based on the number of free electrons, n_e, is difficult. Instead, the code resorts to a subroutine called GENERATELOOKUP which occurs before the MAIN Do loop. GENERATELOOKUP creates lookup tables for values of temperature, free electron density and energy density, creating in effect a data-triplet—given one data point, the other two are determined. Then, with these tables in place, and starting with arbitrary values of energy density, either a corresponding value of temperature or of electron density can be found by interpolating between the values of the appropriate tables. This is exactly what is done after each call to ZAP. The temperature and electron density need periodic updating so ZAP can function properly when it is next called. The program therefore updates TE(J) and DENSITY(J) by calling respectively the subroutines LOOKUPT and LOOKUPN. It is in these subroutines where the interpolations between values of the tables take place. LOOKUPT interpolates between values of energy density, UTABLE,—the X axis—and values of temperatue, TTABLE,—the Y axis. Similarly for

LOOKUPN, where UTABLE is the X axis and the free electron density, NTABLE, is the Y axis.

Starting with the call to GENERATELOOKUP and with the values of χ as input, the subroutine begins by partitioning a range of temperature values from 0.1 eV to 25 eV into 512 evenly spaced time divisions. The program then calculates n_e at each data point by solving the following four simultaneous equations:

$$n_e = n_1 + 2n_2 + 3n_3, \tag{A.7}$$

$$n_e = \frac{n_g}{n_1} a(t), \tag{A.8}$$

$$n_e = \frac{n_1}{n_2} b(t), \tag{A.9}$$

$$n_e = \frac{n_2}{n_3} c(t), \tag{A.10}$$

where

$$a = (2.4E15)t^{\frac{3}{2}} \exp\left(\frac{-\chi_1}{(8.617E-5)t}\right), \tag{A.11}$$

$$b = (2.4E15)t^{\frac{3}{2}} \exp\left(\frac{-\chi_2}{(8.617E-5)t}\right), \tag{A.12}$$

$$c = (2.4E15)t^{\frac{3}{2}} \exp\left(\frac{-\chi_3}{(8.617E-5)t}\right), \tag{A.13}$$

t is the temperature in degrees Kelvin and n_g is the number of ground-state molecules, so that

$$n_g = n_0 - n_1 - n_2 - n_3. \tag{A.14}$$

n_0 equals the number of neutral molecules, n_1 equals the number of singly ionized molecules, n_2 equals the number of doubly ionized molecules, and n_3 equals the number of triply ionized molecules. Equations A.10 to A.12 together with their a, b and c coefficients comprise three SAHA equations, each governing their respective z/z-1 level ratios. We now have four equations in four unknowns.

Solving this system and simplifying leads to a quartic equation in n_e,

$$n_e^4 + (a)n_e^3 + (ab - an_0)n_e^2 + (abc - 2abn_0)n_e - 3abcn_0 = 0. \tag{A.15}$$

For each value of the subroutine's loop counter, TTABLE is loaded based on the above defined partition. Next, Find_ne solves for the positive, real root of the quartic and NTABLE is loaded with that number. From the current NTABLE value, n_1, n_2 and n_3 are solved and from these solutions, the current value of UTABLE is loaded. The result is that all three arrays are loaded at once during each cycle of the loop and hence are coupled, as required by the physics. They are now ready for interpolating.

The portion of the program which creates the tables follows:

```
! Generatelookup is a routine that creates the lookup tables. The three
! values for chi are read in and the program enters a loop, calculating
! values for n1, n2, n3, and ne.  Variables are as follows:
!   chi(3):  an array holding the values of chi
!   ntable:  an array holding the values of n1 for the different temps.
!   n1    :  the instantaneous value of n1
!   n2    :  the instantaneous value of n2
!   n3    :  the instantaneous value of n3
!   a,b,c :  functions of temperature
!   t     :  the temperature in Kelvin
!   tev   :  the temperature in eV
!   ttable:  an array containing the varying temperatures
!   utable:
!   npts  :  number of data points to calculate
!   starttev: starting temperature in eV
!   finaltev: ending temperature in eV
!   tevincr : temperature increment, in eV
!   evtojoule:  constant
!   const :  a constant
!   kev   :  a constant
!   n     :  a constant
!   k     :  a constant
!   i     :  loop variable
!  Something to note is that while the value of ne is recorded for each
!  temperature in ntable, the values of n1, n2, and n3 are not kept for
!  each temperature.
SUBROUTINE generatelookup(n0, chi)
  IMPLICIT none
  REAL*8   chi(3)
  REAL*8   ntable,utable,ttable
  REAL*8   const, kev, tev, starttev, finaltev, tevincr
  REAL*8   evtojoule, t, k, n0
  REAL*8   a, b, c, n1, n2, n3
  COMMON /LINE6/ utable(512), ttable(512), ntable(512)
```

```
  INTEGER i,npts
  PARAMETER(kev = 8.617d-5, evtojoule = 1.602d-19)
  PARAMETER(const = 2.4d15, npts = 512, k = 1.38d-23)
  starttev = .1
  finaltev = 25
  tevincr = (finaltev - starttev)/npts
  tev = starttev
 DO i = 1, npts
   t = tev/kev             ! t in Kelvin!
   a = const*(t**1.5)*dexp(-chi(1)/(t*kev))
   b = const*(t**1.5)*dexp(-chi(2)/(t*kev))
   c = const*(t**1.5)*dexp(-chi(3)/(t*kev))
   ttable(i) = tev
   CALL find_ne(a, b, c, n0, ntable(i))
   n1 = (n0*ntable(i)**2)/((ntable(i)**3)/a + ntable(i)**2 + &
                                       b*ntable(i)+ b*c)
   n2 = b*n1/ntable(i)
   n3 = c*n2/ntable(i)
   utable(i) = evtojoule*(n1*(chi(1) + 3./2.*tev)            &
                       +n2*(chi(1) + chi(2) + 6.*tev)        &
          +n3*(chi(1) + chi(2) + chi(3) + 9./2.*tev))
   tev = tev + tevincr
 END DO         ! end of i loop
RETURN
END SUBROUTINE generatelookup
!---------------------------------------------------------------------
! The subroutine, find_ne, calls zroots to find the 4  roots.
! Its task is to take the three parameters a, b, c and determine
! from them the coefficients of the quadratic equation.  Once
! it receives the 4 roots it eliminates all but the positive, real
! ones and then sends those back. Variables are as follows:
!   a, b, c:  functions of temperature passed to this routine.
!   ntable_real: array holds values of ne returned to the
!   calling routine.  They will be positive and real, hence the
!   name ntable_real.
!   ntable: the values of n1 returned by zroots.
!   Some of these may be complex or negative.
!   coef: an array that holds the values of the coefficients of
!   the quadratic equation.  coef(1) is the constant term, coef(2)
!   is the coefficient of the 1st order term, etc.
!   n0 : a constant also used in determining the values of ne
SUBROUTINE find_ne(a, b, c, n0, ntable_real)
 REAL*8       a, b, c, n0  !used to determine the coefficients
 REAL*8       ntable_real  !the real roots
 COMPLEX*16   coef(5)      !the coefficients
```

```
 COMPLEX*16   netable(4)   !array holding roots returned from
                           !zroots
 coef(1) = dcmplx(-3*a*b*c*n0)
 coef(2) = dcmplx(a*b*c - 2*a*b*n0)
 coef(3) = dcmplx(a*b - a*n0)
 coef(4) = dcmplx(a)
 coef(5) = dcmplx(1)
 CALL zroots(coef, 4, netable, .false.)
!FILTER OUT ONLY VALUES OF Ne WHICH ARE POSITIVE AND REAL
 DO i = 1, 4
  IF (imag(netable(i)) == 0 .AND. real(netable(i)) > 0)      &
                        ntable_real = dreal(netable(i))
 END DO
 RETURN
END SUBROUTINE find_ne
!---------------------------------------------------------------
! The subroutine, zroots, is a driver subroutine.
! It calls LAGUER to find the roots of a polynomial equation.
! m is the degree of the equation, and a is an array containing
! the m+1 coefficients (m coefficients and 1 const. term).
! a(1) =  the constant term, a(2) is the coefficient of x^1, etc.
! "roots"= array containing all the roots, complex or otherwise.
! "polish" = logical variable that determines if the roots will
! be polished infinitely well (true) or only to the desired
! precision (false).  The desired fractional precision is set
! by eps.  "maxm" is the maximum degree the equation can have.
! zroots does the following: it calls laguer multiple times to
! find all the roots of the equation; then it sorts the roots
! for the real one.
SUBROUTINE zroots(a, m, roots, polish)
 IMPLICIT none
 REAL*8 eps
 COMPLEX*16 a(10), roots(4), ad(10), x, b, c, root
 INTEGER i,j,jj,m,maxm
 LOGICAL polish
 PARAMETER (eps=1.E-3, maxm=101)
 DO 11 j = 1, m+1
      ad(j) = a(j)
 11 CONTINUE
 DO 13 j = m, 1, -1
        x = dcmplx(0.0, 0.0)
 CALL laguer(ad, j, x, eps, .false.)
 IF (abs(imag(x)) <= 2.0*eps**2*abs(dreal(x)))     &
        x = dcmplx(dreal(x), 0)
        roots(j)= x
```

```
        b = ad(j+1)
 DO 12 jj = j, 1, -1
        c= ad(jj)
        ad(jj) = b
        b = x*b + c
 12 CONTINUE
 13 CONTINUE
 IF (polish) THEN
 DO 14 j= 1,m
        root = roots(j)
        CALL laguer(a,m,root, eps,.true.)
 14 CONTINUE
 ENDIF
 RETURN
END SUBROUTINE zroots
```

Appendix B. The Timestep Calculation

B.1 Factors Influencing Choice of Time Step

Several factors influenced the choice of time step for the simulation. The first might be called an operational factor, since the decision depended more or less on trial and error. But, rather than being completely arbitrary, the original width used in the old code seemed a reasonable starting point. The second factor involved the shape of the input pulse. If the pulse were linear (a ramp or a square-shaped pulse, for example) equal time steps would be appropriate since the pulse maintains a linear profile throughout its history enforcing a democratic partition: no part of the input can afford to be missed. The shape of the input pulse to the present computer model, however, is gaussian. This shape was chosen because it can be generated relatively easily in a separate computer program. A real input, however, will be a Q-switched pulse (see Figure B.1). In that case, any given layer will experience a rapidly expanding plasma at its beginning with a more moderately changing response near the end of its lifetime. The rapidly varying interaction dynamics at the beginning of the pulse dictates smaller time steps there. The interaction, however, tends to equilibrate after only a few ps, thereby allowing a more coarse sampling rate as time proceeds. The nature of the algorithm, however, prevented using a truly nonlinear partition scheme. As the computation progressed through more and more layers, it became apparent that the time bins at later times were being "compressed" toward the origin before the simulation finished. Consequently, it was decided to divide the pulse into evenly spaced time domains and have each successive time domain have fewer total time bins. This increased the number from 2,500 to over 10,000. The efficiency of the new program, however, has not been degraded appreciably.

The third factor in the choice of time step involved the physics of plasma initiation itself. After approximatly 600 ps, the plasma quits diffusing and the response mechanism isn't changing very rapidly. This further validates the need to rapidly sample the pulse within the first few nanoseconds after plasma ignition.

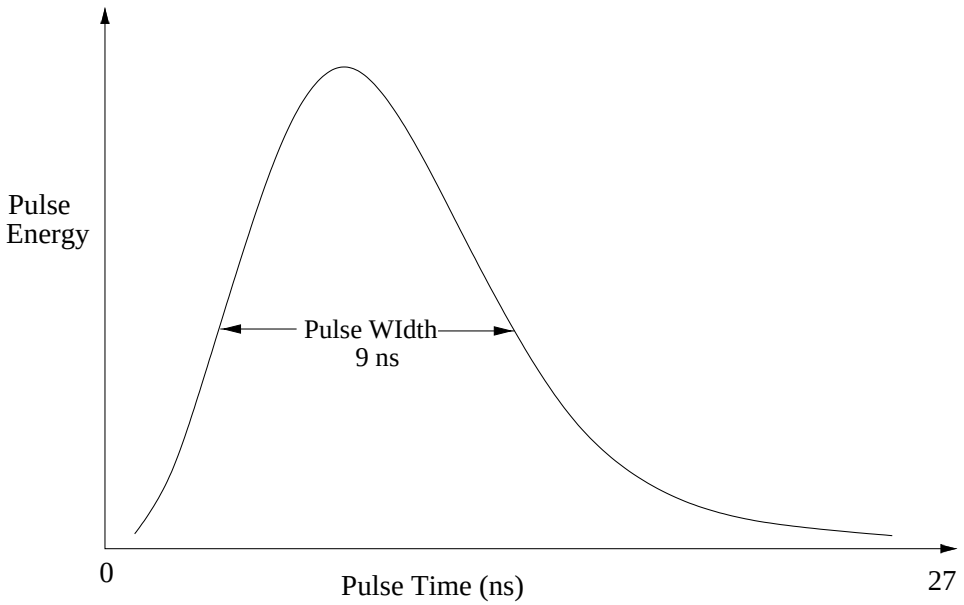

Figure B.1 The sharp rise time of the input Q-switched laser pulse dictates a nonlinear time partition at the beginning to capture and sustain the critical physics of scattering and absorption.

B.2 Estimation of Initial Time Step

With the above factors in mind, we can then estimate the size of the first time step. The governing equation is

$$r^2 = 6Dt, \tag{B.1}$$

where r is the plasma radius and D is the diffusion coefficient (assumed here to be constant). In the computer code, a is the plasma radius and a_0 is the plasma radius at the initial time.

Next, taking the derivative, we have

$$2r\Delta r = 6D\Delta t, \tag{B.2}$$

or

$$\Delta r = \sqrt{\frac{3D}{2t}}\Delta t. \tag{B.3}$$

To capture the essential plasma physics, it is desirable to have $\Delta r \ll r$. (See Figure B.2).

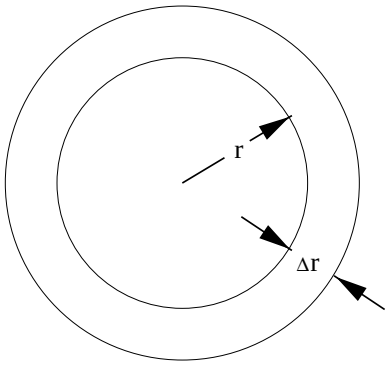

Figure B.2 In the first time step, the plasma expands from r to $r + \Delta r$.

The plasma must "experience" input at a very high rate during this initial expansion phase lest it cool below its critical, self-sustaining temperature. In effect, the wait time for the next input should not exceed approximately 1 ps, which has been determined operationally through experience with the older code.

The effective plasma velocity, V_{eff}, is

$$V_{eff} = \frac{\Delta r}{\Delta t} = \frac{3D}{r}. \tag{B.4}$$

We need to choose Δt such that $\Delta r \ll a_0$. We have $a_0 \sim 20$ nm $= 20 \times 10^{-7}$ cm. Select $\Delta r < 2$ nm $= 2 \times 10^{-7}$ cm or r_1 (for the first time step) $< 20 + 2 = 22$ nm $= 2.2 \times 10^{-7}$ cm. Then

$$(22 \times 10^{-7})^2 = 6Dt_1, \tag{B.5}$$

where D is approximately 0.7 cm^2/sec. Now,

$$\frac{\Delta r}{\Delta t} = \frac{3D}{r}, \tag{B.6}$$

or

$$\Delta t_1 = \frac{40 \times 10^{-14}}{2.1} = 2 \times 10^{-13}. \tag{B.7}$$

Therefore, pick $\Delta t_1 < 0.2$ ps. Choose 0.1 ps as the first time step.

Appendix C. Partial Code Listing

The following is a partial listing of the FORTRAN simulation code used for this thesis project and contains the code for the "A", "B" and "C" loops as well as the code for the ZAP subroutine. The "USE" construction is a Fortran 90 feature which allows the main program to "see" the contents of a separate module, in this case the module "PZ_DATA," which follows immediately below.

```
MODULE PZ_DATA

INTEGER,PARAMETER::REAL8=SELECTED_REAL_KIND(15,307)
INTEGER,PARAMETER::N=20!NUMBER OF LAYERS SAMPLED.
INTEGER,PARAMETER::P=20!NUMBER OF PROCESSORS.

REAL(REAL8),PARAMETER::LL=1000E-4 !SAMPLE LENG(CMS).
REAL(REAL8),PARAMETER::LAMBDA=532E-7!WAVLNGTH(CMS).
REAL(REAL8),PARAMETER::DELTA_L=50E-4!IN CMS(50 MICRONS).
REAL(REAL8),PARAMETER::FNOIN=5. !INPUT F/# OF OPTICS.
REAL(REAL8),PARAMETER::PI=3.141592
REAL(REAL8),PARAMETER::n1=1.36
REAL(REAL8),PARAMETER::SPOT_SIZE=5.*FNOIN*LAMBDA/PI
REAL(REAL8),PARAMETER::RAY_RANGE=4.*LAMBDA*(FNOIN**2)/PI
REAL(REAL8),PARAMETER::bb=LL/FNOIN
REAL(REAL8),PARAMETER::BM_RAD=bb/2.!INPUT BEAM RADIUS.

TYPE POLAR
   REAL(REAL8)::LENGTH
   REAL(REAL8)::ANGLE
END TYPE POLAR
TYPE(POLAR),DIMENSION(0:P,N)::GRID

REAL(REAL8)::DELTA,TEM,LENGTH
REAL(REAL8),DIMENSION(0:30)::LENG,AREA,RAD,HH

!LENG()HOLDS ACCUM LENGTH VALUES STARTING AT LEFT.
!AREA()HOLDS AREA OF EACH LAYER.
!RAD()HOLDS RADIUS OF EACH LAYER.
!DEL()HOLDS DIST BETWEEN THE LAYERS IN RAY_RANGE.
!HH()HOLDS DIST BETWEEN LAYERS THROUGHOUT SAMPLE.

END MODULE PZ_DATA
```

The main program follows next.

```
PROGRAM ss313

USE PZ_DATA

IMPLICIT NONE

!*****************************************************
!  MUST ADD 0.1 J TO THE INTEGRATED INPUT AND OUTPUT
!  FLUENCES.
!*****************************************************

REAL*8  a0, absu, aratio, asratio, bremss, begin
REAL*8  cangle, choice, chi(3), chiic, crit
REAL*8  dconst,diff,e1, endtime, epulse,ethamu, evtojoule
REAL*8  F1, Fc, Fj, fnoout, Ftot
REAL*8  gamma, h, home_stretch, hyp
REAL*8  I1, initdens, initenergy, inittemp, initu
REAL*8  kb, kev
REAL*8  np, nuc,n0c, n0e, n0n, nucconst
REAL*8  plttime,pulsewidth, p0,q,rr,rrr
REAL*8  start, TH, THETA, thetamax,time,timex,t0, tev, tps
REAL*8  u0, width, x, xx, yy, zz, zr
REAL*8  FRAC1, FRAC2,FO_XX,FO_YY,FO_ZZ
REAL*8  TK1,TK2,TQ1,TQ2

INTEGER  I, J, JJ, K, KK, L, NUM
INTEGER  R, S, V, COUNT
INTEGER  ENDK
INTEGER  npts, rpts
INTEGER  tstop
INTEGER  KQ,KKQ
INTEGER  KSUM

LOGICAL  FLAG

CHARACTER(LEN=40) bubble_file, pulse_file
!-----------------------------------------------------------
!      ARRAYS THAT ARE A FUNCTION OF INDEX J (layer)
!-----------------------------------------------------------
REAL*8  dia(31),deltaz(31),zaxis(31)
REAL*8  Ij(31),tau(31),te(31),FRAC(31)
REAL*8  az(31),density(31),energyz(31)
REAL*8  solidw(31),update(31),uz(31)

INTEGER, DIMENSION(0:N) ::  u
```

```
 INTEGER, DIMENSION(30)  ::  U2
 INTEGER, DIMENSION(30)  ::  UU2

 REAL*8, DIMENSION(3,3)  ::  SCAT_MAT
 !---------------------------------------------------------
 !    ARRAYS THAT ARE A FUNCTION OF INDEX K (time)
 !---------------------------------------------------------
 REAL*8 del(10500), deltat(10500), Fout(10500)
 REAL*8 Ifocus(10500), Pinc(10500)
 REAL*8 t(10500), rbubr(10500)
 REAL*8 tb(10500), rbub(10500)
 REAL*8 Pshape(10500), temp(10500)
 !---------------------------------------------------------
 !      ARRAYS THAT ARE A FUNCTION OF I(intensity)
 !---------------------------------------------------------
 REAL*8 ntable(512), ttable(512), utable(512)
 !---------------------------------------------------------
 !          ARRAYS THAT ARE A FUNCTION OF J, K
 !---------------------------------------------------------
 REAL*8, DIMENSION(31,10500) :: Fin, Isfocus, ITEMP
 REAL*8, DIMENSION(31,10500) :: PPSHAPE
 REAL*8, DIMENSION(31,10500) :: RRBUBR
 REAL*8, DIMENSION(31,10500) :: muiso,muplas
 !---------------------------------------------------------
 !             DEFINING DERIVED DATA TYPES
 !---------------------------------------------------------
 TYPE energy
REAL*8 :: value
REAL*8 :: time
 END TYPE energy
 TYPE(energy),DIMENSION(30,10500)  :: Iin, IO, FO
 !---------------------------------------------------------
 !              FUNCTIONS USED BY MAIN PROGRAM
 !---------------------------------------------------------
 REAL*8 :: INTERP, LOOKUPN, LOOKUPT
 !---------------------------------------------------------
 !                    COMMON BLOCKS
 !---------------------------------------------------------
 COMMON /LINE1/ a0,bremss,crit,dconst,ethamu,h,I1,kev,nuc
 COMMON /LINE2/ nucconst,n0n,plttime,q,thetamax,time,x
 COMMON /LINE3/ az,density,deltaz,dia,energyz
 COMMON /LINE4/ solidw,tau,te,uz
 COMMON /LINE5/ muplas,muiso,rrbubr
 COMMON /LINE6/ utable,ttable,ntable
 COMMON /LINE7/ np
```

```
COMMON /LINE8/ deltat,rbubr
!-------------------------------------------------------
!                     PARAMETERS
!-------------------------------------------------------
!noe = initial electron density; kb is Boltzman's constant
PARAMETER(t0 = 5.9d3) !vapor pt of seed mat'l;may + a little
PARAMETER(tps = 2499, tstop = 50)
PARAMETER(n0c = 1d22, n0e = 1.2d20, kb = 1.38d-23)
PARAMETER(evtojoule = 1.602d-19)!const for conv of eV to J
!-------------------------------------------------------
!             INITIAL VALUES OF CONSTANTS
!-------------------------------------------------------
    chiic = 4.3        !work function for seed material
                       !may no longer be used
                       !(used to get initial energy density)
    p0 = 1d9
    pulsewidth = 10d-9 !laser pulse width
    npts = 512
    start = 0          !hvp
    kev = 8.617d-5     !conversion constant from eV to Kelvin
    e1 = n1**2         !Re[dielectric constant] of liquid
    ethamu = 46.0      !amu of ethanol; change for each liquid
    n0n = 1d22         !number density of liquid molecules
    nucconst = 1.8d15  !const to get electron collision fre'cy
!-------------------------------------------------------
!                  FORMAT STATEMENTS
!-------------------------------------------------------
6    FORMAT(f8.3,',',f8.3)
8    FORMAT(a40)
9    FORMAT(f8.3,',',3e20.7)
10   FORMAT(a10)
15   FORMAT(a35,f15.4)
18   FORMAT(a35,e15.4)
17   FORMAT(a35,i15)
12   FORMAT(1x,3e20.7)
28   FORMAT(1x,'Pulse width     =',a4,' ns')
29   FORMAT(a10,1e10.3)
31   FORMAT(a10,a2,f10.2)
41   FORMAT(1x,'Enter your choice:',i3,',',i3,','' or 0 for same')
51   FORMAT(' ',A3,1X,G12.6,1X,G12.6,1X,G12.6,1x,G12.6,1x,G12.6,1X,G12.6,1X,G12.6)
53   FORMAT(' ',A3,1X,G12.6,1X,G12.6,1X,G12.6)
120  FORMAT(1x,' density = ',1e19.5)
121  FORMAT(1x,' density = ',1e19.5,' Transmission = ',1e19.5)
122  FORMAT(1x,2e20.10)
123  FORMAT(1x,3e20.10)
```

```
 130  FORMAT(1x,'Calculating Input Fluence : ',f10.3,' J/cm^2')
OPEN(unit = 81, file = 'den1.dat', status='unknown')
!-------------------------------------------------------------
!                INITIAL SAVED CONSTANTS
!-------------------------------------------------------------
!    a0 = particle size
! gamma = ratio of heat cap at const vol to hc at const press
! fnoin = the f/number of the input optics
!fnoout = the f/number of the output optics
!    np = initial particle density
!dconst = diffusion constant
!  mult = constant
!     h = constant in diffusion algorithm
! width = laser pulse width
!chi(n) = ionization potentials of the liquid
!epulse = laser pulse energy

 WRITE(*,*) 'Single Shot Version: press RETURN to continue.'
 READ(*,*)

OPEN(unit = 10, file = 'ssdata.dat', status = 'old')
!-------------------------------------------------------------
!                   READ IN CONSTANTS
!-------------------------------------------------------------
        REWIND(10)
        READ(10,*) a0
        READ(10,*) gamma
        READ(10,*) fnoout
        READ(10,*) np
        READ(10,*) dconst
        READ(10,*) h
        READ(10,*) x
        READ(10,*) width
        READ(10,*) chi(1)
        READ(10,*) chi(2)
        READ(10,*) chi(3)
        READ(10,*) epulse
        CLOSE(10)
5 CONTINUE
!-------------------------------------------------------------
!             DISPLAY CONSTANTS TO SCREEN
!-------------------------------------------------------------
        WRITE(*,*) '    a0 = ',a0
        WRITE(*,*) ' gamma = ',gamma
        WRITE(*,*) 'fnoout = ',fnoout
```

```
        WRITE(*,*) '    np = ',np
        WRITE(*,*) 'dconst = ',dconst
        WRITE(*,*) '     h = ',h
        WRITE(*,*) '     x = ',x
        WRITE(*,*) ' width = ',width
        WRITE(*,*) 'chi(1) = ',chi(1)
        WRITE(*,*) 'chi(2) = ',chi(2)
        WRITE(*,*) 'chi(3) = ',chi(3)
        WRITE(*,*) 'epulse = ',epulse

 WRITE(*,*)
 WRITE(*,*) 'The input numbers: press RETURN to continue.'
 READ(*,*)
!-------------------------------------------------------------
!     CALCULATING THE NONLINEAR TIME PARTITION
!      ENDTIME = 2.72711E-08 ns.
!-------------------------------------------------------------
!    DO K = 1, 2000
!        DELTAT(K) = 1.00E-14
!    ENDDO
!    DO K = 2001, 3000
!        DELTAT(K) = 1.50E-13
!    ENDDO
!    DO K = 3001, 4000
!        DELTAT(K) = 3.50E-13
!    ENDDO
!    DO K = 4001, 5000
!        DELTAT(K) = 7.50E-13
!    ENDDO
!    DO K = 5001, 6000
!        DELTAT(K) = 2.00E-12
!    ENDDO
!    DO K = 6001, 7000
!        DELTAT(K) = 3.50E-12
!    ENDDO
!    DO K = 7001, 8000
!        DELTAT(K) = 5.50E-12
!    ENDDO
!    DO K = 8001, 9000
!        DELTAT(K) = 7.50E-12
!    ENDDO
!    DO K = 9001, 10000
!        DELTAT(K) = 9.00E-12
!    ENDDO
!    TIME = 0.0
```

```
!    DO K = 1, 10000
!        TIME = TIME + DELTAT(K)
! write(*,*)'ttt',k,deltat(k),del(k),time
! write(*,*)'ttt',k,deltat(k)*1E+10,del(k)*1E+10,time
!   IF(TIME > ENDTIME)EXIT
!   ENDDO
  KSUM = 10000
!   WRITE(*,*) 'Calculated time and ENDTIME:',TIME,ENDTIME
!-------------------------------------------------------------
!SELECTING THE INPUT FILE CONTAINING THE ENERGY
! OF THE INCIDENT PULSE
!-------------------------------------------------------------
        l = 1
        IF (width.eq.9) THEN
           pulse_file = 'gaussinput_10000'
!          pulse_file = 'qs9.25t'
!          pulse_file = 'const_temp_k.in'
        ELSEIF (width .eq. 35) THEN
           pulse_file = 'PULSE35.035'
        ELSEIF (width .eq. 50) THEN
           pulse_file = 'ps50n.dat'
        ELSE
           WRITE(*,*) 'pulse shape not available for width'
       STOP
        END IF
!-------------------------------------------------------------
!       FILLING THE TEMP AND PSHAPE ARRAYS.
!       DEL AND PINC ARRAYS ARE FOR TAU CALCUATION LATER.
!-------------------------------------------------------------
        TEMP(1) = 0.0
        DELTAT(1)  = 1.0D-14
!       DEL(1)  = 2.3D-14
        OPEN(UNIT = 14, FILE = pulse_file, STATUS = 'old')
        REWIND(14)
        READ(14,*) NUM
        DO K = 1, NUM
            READ(14,*) TEMP(K), PSHAPE(K)
            IF(K > 1) DELTAT(K) = TEMP(K) - TEMP(K-1)
PINC(K) = EPULSE * PSHAPE(K)
        ENDDO
        CLOSE(14)
        ENDK = K - 1
!       ENDTIME = TEMP(ENDK)
        ENDTIME = 2.72711E-08!value taken from file qs9.25t.
!-------------------------------------------------------------
```

```
!                 EXPLANATION OF GRIDFILL
!-------------------------------------------------------
!GRIDFILL CONTAINS THE MECHANICS FOR CALCULATING THE RADIUS
!AND AREA OF EACH PRIMARY ZONE (PZ) LAYER AND THEIR INTERVEN-
!ING DISTANCES AND THE POLAR COORDINATES OF THE TRIANGULAR
!GRID WHICH DISCRETIZES THE PZ. J COUNTS THE LAYERS STARTING
!FROM NO 1 ON THE RIGHT EDGE.

          CALL GRIDFILL

!FILL A 3 X 3 ARRAY WITH NUMBERS THAT MIMIC SCATTERING. THE
!ARRAY'S ENTRIES CORRESPOND TO NINE PAIR COMBINATIONS WHICH
!RESULT FROM THREE INPUTS MAPPED TO THREE OUTPUTS. RESHAPE
!IS A FORTRAN 90 INTRINSIC. SCATTERING NUMBERS IN THE ARRAY
!ARE HARDWIRED FOR NOW...
!-------------------------------------------------------
!       CALUCULATING THE SOLID ANGLE FOR EACH LAYER.
!-------------------------------------------------------
!SOLIDW =  array holding values of solid angles--layer depend.
!DIA = the diameter of each layer, computed from RAD(J)
!CANGLE = the value of the cosine of the linear angle
!HYP = the hypotenuse of the triangle

  THETAMAX = ASIN(1./(2.*fnoout)) !maximum collection angle!

!FORMULA FOR SOLIDW(N) DERIVED USING SERIES APPROXIMATION
!FOR 1/(1+X) WHERE X = TAN^2(X) BECAUSE COS^2(X) = 1/(1+TAN^2(X)
!AND TAN(X) EQUALS 1/2*f/#OUT. THE SOLID ANGLE FOR EACH SUBSEQ
!LAYER USES 2*PI*(1-COS(theta)) WHERE COS(theta) IS FOUND MORE
!DIRECTLY USING HH AND HYP OF THE LINEAR ANGLE BEWTEEN LAYERS.
!HH() ARRAY HAS ALREADY BEEN CALC IN PZ DATA AND HOLDS DIST
!BETWEEN LAYERS.

!SOLIDW(1) = 2.*PI*(1-SQRT(1-.25/fnoout**2+1./(16.*fnoout**4)))
 DIA(1)    = 2. * RAD(1)
    DO J = 2, N
       DIA(J)    = 2. * RAD(J)
!      HYP       = SQRT((HH(J+1)**2)+(RAD(J+1)**2))
       HYP       = SQRT((HH(J)**2)+(RAD(J)**2))
!      CANGLE    = HH(J+1)/HYP
       CANGLE    = HH(J)/HYP
       SOLIDW(J) = 2.*PI*(1.- CANGLE)
    END DO
 SOLIDW(1) = SOLIDW(2)
!-------------------------------------------------------
```

```
!       INITIAL ENERGY DENSITY AT PLASMA INITIATION
!       T0 IS INITIAL TEMP; U0 IS ENERGY DENSITY.
!-------------------------------------------------------
U0 = (CHIIC + (3./2.)*KEV*T0)*N0E*EVTOJOULE
INITENERGY = U0*4./3.*PI*A0**3 !ENERGY IN EACH PLASMA CENTER.
!-------------------------------------------------------
!                    SAHA TABLES
!-------------------------------------------------------
  DO I = 1, NPTS                !NPTS = 512
     UTABLE(I) = 0.0
     NTABLE(I) = 0.0
     TTABLE(I) = 0.0
  END DO

  CALL GENERATELOOKUP(1D22, CHI)

  INITU       =   (CHIIC+3./2.*KEV*T0)*N0E*EVTOJOULE
  INITTEMP    =   T0
  INITDENS    =   7.0D18
  WRITE(*,*) 'Init Temperature =',INITTEMP
  WRITE(*,*) 'Init Density     =',INITDENS
!-------------------------------------------------------
! FILLING BUBBLE TIME AND BUBBLE GROWTH ARRAYS, TB() AND RBUB().
!-------------------------------------------------------
   bubble_file = 'bub80c3.dat'
   OPEN(UNIT = 7, FILE = bubble_file, STATUS = 'old')
   REWIND(7)
   I = 1
   READ(7,*) NUM
   DO WHILE (I <= NUM)
      READ(7,*) TB(I), RBUB(I)
      I = I + 1
   ENDDO
   CLOSE(7)
   RPTS = I - 1
!-------------------------------------------------------
!                SCATTERING MATRIX
!-------------------------------------------------------

SCAT_MAT = RESHAPE(SOURCE=(/.10,.30,.60,.25,.75,.25,.60,.30,   &
.10/),  SHAPE = (/3,3/))

!-------------------------------------------------------
!                    CALCULATING TAU
!-------------------------------------------------------
```

```
!EACH LAYER NEEDS TO HAVE ASSIGNED TO IT ITS OWN TAU, THE TIME
!TO REACH ITS CRITICAL FLUENCE ,FC, WHICH IS SET AT 0.1 JOULE.
DO J = N, 1, -1
    FTOT = 0.0
    FC   = 0.1  !FC = PROPERTY OF CARBON; SAME FOR EACH LAYER.
!   TIME = 0.0
    K = 1
!NEXT DO LOOP CRUDELY INTEGRATES INTENSITY. IJ HOLDS ITENSITY
!FOR EACH LAYER. FJ UPDATES FTOT FOR EACH COUNT OF K.
    DO WHILE(FTOT < FC) !WHEN TOTAL FLUENCE = FC, EXIT DO LOOP.
        IJ(J) = PINC(K)/AREA(J)
        FJ = IJ(J) * DELTAT(K)
        FTOT = FTOT + FJ
! TIME = TIME + DELTAT(K)
        IF( K < ENDK ) K = K + 1
    ENDDO
 TAU(J) = TEMP(K) !TEMP(K) HOLDS TIME VALUE,TAU(J),
 UU2(J) = K   !FOR LAYER J TO REACH ITS CRITICAL
                      !FLUENCE.
ENDDO
!-------------------------------------------------------------
!AA LOOP FILLS EACH LAYER'S ARRAY WITH INTERPOLATED VALUES.
!-------------------------------------------------------------
AA: DO J = N, 1, -1
       TIME = TAU(J)
       FLAG = .TRUE.
IF(J == N)THEN
    FIRST:DO I = UU2(J), KSUM
    PPSHAPE(J,I) = Pshape(I)
ISFOCUS(J,I) = Pinc(I)/AREA(1)
U2(J) = I
TIME = TIME + DELTAT(I)
  IF(TIME > ENDTIME)EXIT
END DO FIRST
ELSE IF( J .NE. N)THEN
    SECOND:DO I = UU2(J), KSUM
PPSHAPE(J,I) = Pshape(I)
ISFOCUS(J,I) = Pinc(I)/AREA(1)
IF(TIME > TAU(J+1) .AND. FLAG)THEN
U2(J) = I - 1
FLAG  = .FALSE.
END IF
TIME = TIME + DELTAT(I)
IF(TIME > ENDTIME)EXIT
END DO SECOND
```

```
END IF
END DO AA
!-------------------------------------------------------------
!       REINITIALIZE ARRAYS AT THE BEGINNING OF EACH SHOT.
!-------------------------------------------------------------
 TIME = 0.0
 DO J = 1, N
     UZ(J)       =   UO
     ENERGYZ(J)  =   INITENERGY
     AZ(J)       =   AO
     TE(J)       =   INITTEMP    !INITIAL TEMPERATURE
     DENSITY(J)  =   INITDENS    !INITIAL ELECTRON DENSITY
 END DO
!-------------------------------------------------------------
!      REINITIALIZE CONSTANTS AT THE BEGINNING OF EACH SHOT.
!-------------------------------------------------------------
  TEV     =   TE(J-1)*KEV                !TEMP IN eV
  DIFF    =   DCONST*(SQRT(TEV)+X*TEV**H)!CALC DIFF CONSTANT
  CRIT    =   (3./(4.*PI*NP))**(1./3.)  !PLMA VOL = LATT VOL.
  NUC     =   NUCCONST
!-------------------------------------------------------------
!   MAIN LOOP OF REVISED TIME AND LAYER ACTIVATION SCHEME.
!-------------------------------------------------------------
    OPEN(unit = 43, file = 'results', status = 'OLD')
!-------------------------------------------------------------
 MAIN: DO J = N, 1, -1
!-----ADJUSTING TIME SCALE FOR THE BUBBLE FILE--------------
!-----------------WITH EACH NEW J---------------------------
 TIME = 0.0
 K = 1
 DO KK = UU2(J), KSUM !KSUM = 10000
  R = 2
  DO WHILE ((R < RPTS) .AND. (TB(R) <= TIME))
   R = R + 1
  END DO
  IF(R <= RPTS)THEN
   RBUBR(K) = INTERP(RBUB(R-1),RBUB(R),TB(R-1),TIME,TB(R))
  ELSE
   RBUBR(K) = RBUBR(K-1)
   END IF
   TIME = TIME + DELTAT(KK)
   K = K + 1
 END DO
!-------------------------------------------------------------
K = 1
```

```
TIME = TAU(J)
ASRATIO = AREA(1)/AREA(J)
ARATIO  = AREA(J+1)/AREA(J)
!-------------------------------------------------------
IF(J == N)THEN   !FIRST LAYER IS UNIQUE: NO NEIGHBOR INPUTS.
!-------------------------------------------------------
!------------LOOP "A" FOR LAYER 20 ONLY--------------------
A: DO I = UU2(J), KSUM   !I indexes into some point in the
!pulse profile. K must start at one when layer turns on.
 K = I - UU2(J) + 1
 I1 = ISFOCUS(J,I)*ASRATIO
 F1 = I1*DELTAT(I)
 PLTTIME = TIME * 1E+09
     CALL ZAP(J,K,F1)
 IO(J,I)%value = I1*DEXP(-MUPLAS(J,K)*HH(J))     &
*DEXP(-MUISO(J,K)*HH(J))
 IO(J,I)%time  = TIME
 Iin(J-1,I)%value = IO(J,I)%value
 Iin(J-1,I)%time  = IO(J,I)%time
 TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
 DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
 FO(J,I)%value = IO(J,I)%value*DELTAT(I)
 FO(J,I)%time  = IO(J,I)%time
 IF(FO(J,I)%value < 1E-13) FO(J,I)%value = 0.00
    TIME = TIME + DELTAT(I)
    IF(TIME > ENDTIME)EXIT
END DO A
!-------------------------------------------------------
ELSE IF (J < N .AND. J > 1) THEN
!-------------------------------------------------------
    TQ1 = TAU(J)     !TQ2-TQ1="PROBE INTVL" : LAYER J.
    TQ2 = TQ1 + DELTAT(1)
!---------------LOOP "B"---------------------------------
B: DO KQ = UU2(J), UU2(J+1)
 K = KQ - UU2(J) + 1
 I1 = ISFOCUS(J,KQ)*ASRATIO
 F1 = I1*DELTAT(KQ)
 PLTTIME = TIME * 1E+09
     CALL ZAP(J,K,F1)
 IO(J,KQ)%value = I1*DEXP(-MUPLAS(J,K)*HH(J))     &
*DEXP(-MUISO(J,K)*HH(J))
 IO(J,KQ)%time = TIME
 Iin(J-1,KQ)%value = IO(J,KQ)%value
 Iin(J-1,KQ)%time = IO(J,KQ)%time
 TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
```

```
 DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
 FO(J,KQ)%value = IO(J,KQ)%value*DELTAT(KQ)
 FO(J,KQ)%time = IO(J,KQ)%time
!-----------------------------------------------------
  IF(KQ == UU2(J+1))THEN
!-----------------------------------------------------
 XX = TAU(J+1)-TQ1 !XX IS THE LAST BIT OF "FREE" INPUT INTO LAYER J.
 I1 = ISFOCUS(J,KQ)*ASRATIO
 F1 = I1*XX
     CALL ZAP(J,K,F1)
 IO(J,KQ)%value = I1*DEXP(-MUPLAS(J,K)*HH(J))     &
      *DEXP(-MUISO(J,K)*HH(J))
 IO(J,KQ)%time = TIME
 Iin(J-1,KQ)%value = IO(J,KQ)%value
 Iin(J-1,KQ)%time = IO(J,KQ)%time
 TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
 DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
 FO(J,KQ)%value = IO(J,KQ)%value*XX
 FO(J,KQ)%time = IO(J,KQ)%time
!-----------------------------------------------------
  END IF
!-----------------------------------------------------
 TQ1 = TQ2
 TQ2 = TQ2 + DELTAT(KQ+1)
 TIME = TQ1
 PLTTIME = TIME*1E+09
END DO B
!-----------------------------------------------------
 KK = UU2(J+1) !KK = Layer J+1's time counter.
         TK1 = TAU(J+1)
         TK2 = TK1 + DELTAT(KK)
!----------------LOOP "C"----------------------------
C: DO KQ = UU2(J+1), KSUM
!-----------------------------------------------------
 PLTTIME = TIME * 1E+09
 DO
  IF(TK1 < TQ2 .AND. TK2 > TQ2)EXIT
  Fin(J,KQ) = FO(J+1,KK)%value*ARATIO
    CALL ZAP(J,K,Fin(J,KQ))
  IO(J,KQ)%value = Iin(J,KK)%value*ARATIO        &
*DEXP(-MUPLAS(J,K)*HH(J))    &
*DEXP(-MUISO(J,K)*HH(J))
  IO(J,KQ)%time = Iin(J,KK)%time
  FO(J,KQ)%value = FO(J,KQ)%value + IO(J,KQ)%value*DELTAT(KK)
  FO(J,KQ)%time =  Iin(J,KK)%time
```

```
  Iin(J-1,KQ)%value = IO(J,KQ)%value
  Iin(J-1,KQ)%time = IO(J,KQ)%time
  TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
  DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
  TK1 = TK2
  TK2 = TK2 + DELTAT(KK+1)
  IF(TK1 < TQ2 .AND. TK2 > TQ2)EXIT
  K = K + 1
  KK = KK + 1
 END DO
  XX = TQ2 - TK1
  YY = TK2 - TQ2
  Fin(J,KQ) = ARATIO*FO(J+1,KK)%value*XX/DELTAT(KK)
       CALL ZAP(J,K,Fin(J,KQ))
  IO(J,KQ)%value = Iin(J,KK)%value*ARATIO      &
*DEXP(-MUPLAS(J,K)*HH(J))    &
*DEXP(-MUISO(J,K)*HH(J))
  IO(J,KQ)%time = Iin(J,KK)%time
  FO(J,KQ)%value = FO(J,KQ)%value + IO(J,KQ)%value*XX
  FO(J,KQ)%time = Iin(J,KK)%time
  Iin(J-1,KQ)%value = IO(J,KQ)%value
  Iin(J-1,KQ)%time  = IO(J,KQ)%time
  TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
  DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
  KKQ = KQ + 1
  Fin(J,KKQ) = ARATIO*FO(J+1,KK)%value*YY/DELTAT(KK)
! Fin(J,KKQ) = ARATIO*FO(J+1,KK)%value
       CALL ZAP(J,K,Fin(J,KKQ))
  IO(J,KKQ)%value = Iin(J,KK)%value*ARATIO      &
*DEXP(-MUPLAS(J,K)*HH(J))    &
*DEXP(-MUISO(J,K)*HH(J))
  IO(J,KKQ)%time = Iin(J,KK)%time
  FO(J,KKQ)%value = IO(J,KKQ)%value*YY
  Iin(J-1,KKQ)%value = IO(J,KKQ)%value
  Iin(J-1,KKQ)%time  = IO(J,KKQ)%time
  TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
  DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
  TIME = TIME + DELTAT(KQ)
  IF(TIME > ENDTIME)EXIT
  TQ1 = TQ2
  TQ2 = TQ2 + DELTAT(KQ+1)
  TK1 = TK2
  TK2 = TK2 + DELTAT(KK+1)
  KK = KK + 1
  K = K + 1
```

```
END DO C
!-------------------------------------------------
ELSE IF(J == 1)THEN
!-------------------------------------------------
TQ1 = TAU(J)
TQ2 = TQ1 + DELTAT(1)
ARATIO = AREA(2)/AREA(1)
!------------------LAST LAYER-----------------------
LAST1:DO KQ = UU2(1), UU2(2) !KQ = LAYER 1 TIME COUNTER
     K = KQ - UU2(1) + 1
I1 = ISFOCUS(J,KQ)*ASRATIO
F1 = I1*DELTAT(KQ)
PLTTIME = TIME * 1E+09
    CALL ZAP(J,K,F1)
IO(J,KQ)%value = I1*DEXP(-MUPLAS(J,K)*HH(J))       &
*DEXP(-MUISO(J,K)*HH(J))
IO(J,KQ)%time = TIME
TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
FO(J,KQ)%value = IO(J,KQ)%value*DELTAT(KQ)
FO(J,KQ)%time = IO(J,KQ)%time
   !-----------------------------------------------
   IF(KQ == UU2(2))THEN
   !-----------------------------------------------
     XX = TAU(J+1)-TQ1 !XX IS THE LAST BIT OF "FREE"
       !INPUT INTO LAYER J.
     I1 = ISFOCUS(J,KQ)*ASRATIO
     F1 = I1*XX
          CALL ZAP(J,K,F1)
     IO(J,KQ)%value = I1*DEXP(-MUPLAS(J,K)*HH(J))  &
  *DEXP(-MUISO(J,K)*HH(J))
     IO(J,KQ)%time = TIME
     TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
     DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
     FO(J,KQ)%value = IO(J,KQ)%value*XX
     FO(J,KQ)%time = IO(J,KQ)%time
   !-----------------------------------------------
    END IF
   !-----------------------------------------------
 TQ1 = TQ2
 TQ2 = TQ2 + DELTAT(KQ+1)
 TIME = TQ1
 PLTTIME = TIME * 1E+09
END DO LAST1
!-------------------------------------------------
```

```
 KK = UU2(2)        !KK = LAYER 2 TIME COUNTER.
 TK1 = TAU(2)
 TK2 = TK1 + DELTAT(KK)
 TIME = TK1
!-------------------------------------------------
LAST2: DO  KQ = UU2(2)+1, KSUM
 PLTTIME = TIME * 1E+09
 IF(TK1 < TQ2 .AND. TK2 > TQ2)THEN
   Fin(J,KQ) = FO(J+1,KK)%value*ARATIO
CALL ZAP(J,K,Fin(J,KQ))
   IO(J,KQ)%value =Iin(J,KK)%value*ARATIO  &
  *DEXP(-MUPLAS(J,K)*HH(J))       &
    *DEXP(-MUISO(J,K)*HH(J))
   IO(J,KQ)%time = Iin(J,KK)%time
   FO(J,KQ)%value = FO(J,KQ)%value +  &
IO(J,KQ)%value*DELTAT(KK)
   FO(J,KQ)%time = Iin(J,KK)%time
   TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
   DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
   TK1 = TK2
   TK2 = TK2 + DELTAT(KK+1)
   K = K + 1
   KK = KK + 1
 END IF
 XX = TQ2 - TK1
 YY = TK2 - TQ2
 Fin(J,KQ) = ARATIO*FO(J+1,KK)%value*XX/DELTAT(KK)
!Fin(J,KQ) = ARATIO*FO(J+1,KK)%value
    CALL ZAP(J,K,Fin(J,KQ))
 IO(J,KQ)%value=Iin(J,KK)%value*ARATIO &
     *DEXP(-MUPLAS(J,K)*HH(J))            &
     *DEXP(-MUISO(J,K)*HH(J))
 IO(J,KQ)%time = Iin(J,KK)%time
 FO(J,KQ)%value = FO(J,KQ)%value + IO(J,KQ)%value*XX
 FO(J,KQ)%time = Iin(J,KK)%time
 TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
 DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
 KKQ = KQ + 1
 Fin(J,KKQ) = ARATIO*FO(J+1,KK)%value*YY/DELTAT(KK)
!Fin(J,KKQ) = ARATIO*FO(J+1,KK)%value
      CALL ZAP(J,K,Fin(J,KKQ))
 IO(J,KKQ)%value = Iin(J,KK)%value*ARATIO &
      *DEXP(-MUPLAS(J,K)*HH(J)) &
      *DEXP(-MUISO(J,K)*HH(J))
 IO(J,KKQ)%time = Iin(J,KK)%time
```

```
    FO(J,KKQ)%value = IO(J,KKQ)%value*YY
    FO(J,KQ)%time = Iin(J,KK)%time
    TE(J) = LOOKUPT(UZ(J),UTABLE,TTABLE)
    DENSITY(J) = LOOKUPN(UZ(J),UTABLE,NTABLE)
    TIME = TIME + DELTAT(KQ)
    IF(TIME > ENDTIME)EXIT
    TQ1 = TQ2
    TQ2 = TQ2 + DELTAT(KQ+1)
    TK1 = TK2
    TK2 = TK2 + DELTAT(KK+1)
K = K + 1
    KK = KK + 1
END DO LAST2
!-----------------------------------------------------
END IF
!-----------------------------------------------------
END DO MAIN
!-----------------------------------------------------
J = 1
TIME = TAU(1)
RESULTS:DO  K = UU2(1), KSUM
IF(IO(J,K)%value > 3E+09)IO(J,K)%value = 0.0
WRITE(43,*) K,TIME*1E+09,IO(J,K)%value,Iin(J,K),Fin(J,K)
TIME = TIME + DELTAT(K)
IF(TIME > ENDTIME)EXIT
END DO RESULTS

!Integrate total output intensity to obtain total output
!fluence, adding 0.1 J/cm^2 to allow for plasma initiation

   CLOSE(14)
   CLOSE(43)
   CLOSE(44)
   CLOSE(45)
   CLOSE(81)

WRITE (*,*) 'ss313 completed'

END PROGRAM ss313
!-------------------------ZAP-------------------------------
SUBROUTINE ZAP(jv,ku,fcell)

    USE PZ_DATA
    IMPLICIT none
```

```
      REAL*8 a0, az(31), absu, alpha
      REAL*8 bremss, crit, diff, dconst, deltat(10500)
      REAL*8 deltaz(31), density(31), dia(31)
      REAL*8 enliq, energya, energyz(31), ethamu
      REAL*8 e2p, e2pp, eps2p, eps2pp, evtemp
      REAL*8 fluence, fcell, hold
      REAL*8 h, I1, iext,  kk, kev
      REAL*8 muplas(31,10500), muiso(31,10500)
      REAL*8 n0n, nueff, np, ntot, nuc, nucconst
      REAL*8 ntable(512), plastime, plttime, pwliq
      REAL*8 plta(31), pltu(31)
      REAL*8 q, qa, rb, rrbubr(31,10500), rbubr(10500), sigiso
      REAL*8 sigmaa, sigmasph, solidw(31), stim, stimemiss
      REAL*8 tau(31), te(31), thetamax, time, ttable(512)
      REAL*8 utable(512), uz(31), vol, x

      COMMON /LINE1/ a0,bremss,crit,dconst,ethamu,h,I1,kev,nuc
      COMMON /LINE2/ nucconst,n0n,plttime,q,thetamax,time,x
      COMMON /LINE3/ az,density,deltaz,dia,energyz
      COMMON /LINE4/ solidw,tau,te,uz
      COMMON /LINE5/ muplas,muiso,rrbubr
      COMMON /LINE6/ utable,ttable,ntable
      COMMON /LINE7/ np
      COMMON /LINE8/ deltat,rbubr

      REAL*8, PARAMETER  ::  c = 3d10
      REAL*8, PARAMETER  ::  Emin = 2.4e-19 !min elect NRG in Joules!
      REAL*8, PARAMETER  ::  eamu = (1./1837.)  !electron mass in amu!
      REAL*8, PARAMETER  ::  elp = n1**2
      REAL*8, PARAMETER  ::  b = ((elp - 1.)/(2.*elp + 1.))**2

      INTEGER jv,ku

     rb       = rbubr(ku)
     kk       = 2.*pi/lambda
     evtemp   = te(jv)*kev
     alpha    = kk*rb
     hold     = 8./3.*pi*b*kk**4*rb**6
     nueff    = 2*(eamu/ethamu)*nuc*(evtemp**0.5)
!    plastime = (time - tau(jv) + 1.e-12)
     plastime = (time - tau(jv) + 1.e-13) !ZAP won't run withou a small
!    time increment to get the plasma expansion ball rolling.
!    plastime = (time - tau(jv))
     vol      = (pi/4.)*(dia(jv)**2)*HH(jv)
     ntot     = np*vol
```

```
!CALCULATION OF LIQUID THERMALIZATION

   pwliq = nueff*energyz(jv)
   enliq = pwliq*deltat(ku)

!CALCULATION OF THE BUBBLE SCATTERING, ISOTROPIC AND FRAUNHOFF

         CALL isotro(hold, solidw(jv), sigiso)
         muiso(jv,ku) = sigiso*np

!----ABSORPTION CROSSECTION---(RAYLEIGH REGION)----------------
!----------------------Zapregion1-----------------------------
IF (az(jv) .LT. lambda/2.) THEN
eps2p = e2p(density(jv),nuc,lambda) !Re[plasma dielectric constant]!
eps2pp = e2pp(nuc,lambda,te(jv),density(jv))
                            !Im[plasma dielectric constant]
qa = sigmaa(az(jv),a0,lambda,n1,eps2p,eps2pp,rbubr(ku),jv,ku)
                         !Rayleigh absorption cross section
stim = stimemiss(lambda,te(jv)) !stimulated emission factor
muplas(jv,ku) = np*(qa*stim)
energya = fcell*qa*stim     !absorbed energy per particle
energyz(jv) = energyz(jv)+energya-(enliq/ntot)!-(erad/ntot)
                            !energy per carbon particle
IF (energyz(jv).LE.Emin) energyz(jv) = Emin
diff = dconst*(sqrt(evtemp) + x*evtemp**h)!calc diff constant
az(jv) = a0 + dsqrt(6.*diff*plastime)    !plasma expansion
uz(jv) = energyz(jv)/((4.*pi/3.)*az(jv)**3) !NRG density
                                            !due to expansion
!---------------------Zapregion1-------------------------------
!----INTERMEDIATE REGION-----------------------------------------
!----------------------Zapregion2-------------------------------
ELSE IF((az(jv).GT.lambda/2.).AND.(az(jv).LT.CRit)) THEN
sigmasph=(4./3.)*pi*(az(jv)**3 - rbubr(ku)**3 + a0**3)  &
       *((0.106*(1/n1)*(nuc/((2.*pi*c/lambda)**2        &
       +nuc**2)))*density(jv) + (1.37E-25)*lambda**3    &
       /dsqrt(te(jv)*8.617d-5)*(density(jv))**2)

q = sigmasph

stim=stimemiss(lambda,te(jv)) !stimulated emission factor
muplas(jv,ku)=np*(q*stim)     !plasma  crossection

energya=fcell*q*stim     !absorbed energy per particle
energyz(jv)=energyz(jv)+energya-(enliq/ntot) !-(erad/ntot)
```

```
                          !energy per carbon particle
IF (energyz(jv).LE.Emin) energyz(jv) = Emin
diff=dconst*(sqrt(evtemp)+x*evtemp**h)!calc diff constant!
az(jv)=a0+dsqrt(6.*diff*plastime) !Let plasma center expand!
uz(jv)=energyz(jv)/((4.*pi/3.)*az(jv)**3)
              !energy density due to diffusion of electrons!
!--------------------Zapregion2--------------------------------
!---CONTINUUM REGION--(plasma spheres have coalesced)----------
!--------------------Zapregion3--------------------------------
ELSE IF (az(jv) .GE. crit) THEN
bremss = iext(nuc,lambda,n1,te(jv),density(jv))          &
                    *(1./np - 4.*pi/3.*(rbubr(ku))**3)*np
!Could run into problems when bubble radius exceeds lattice size
stim = stimemiss(lambda,te(jv))  !stimulated emission factor
muplas(jv,ku) = bremss*stim      !plasma crossection!
fluence = fcell*(1. - dexp(-bremss*HH(jv)*stim))
                                          !absorbed fluence!
absu = fluence/HH(jv) !Absorbed energy density!
energyz(jv) = energyz(jv) + absu*vol  - enliq    ! - Erad !
IF (energyz(jv).LE.Emin) energyz(jv) = Emin
uz(jv) = uz(jv) + absu  - enliq/vol   ! - erad/vol !
IF (energyz(jv).LE.Emin) uz(jv) = 5e-19

ELSE
STOP 'something screwy happened with coelescent limits'
!----------------------Zapregion3--------------------------------
ENDIF
plta(jv) = az(jv)*1e7 !convert units for plasma radius to nm!
pltu(jv) = uz(jv)*1e3 !convert units for energy density to mJ/cm3!
RETURN
END SUBROUTINE ZAP
```

Bibliography

1. Brown, S. C. *Basic Data of Plasma Physics, 1966*, second ed. The M.I.T. Press, 1967.

2. Forster, H. K., and Zuber, N. Growth of a vapor bubble in a superheated liquid. *Journal of Applied Physics 25*, 4 (1954), 474–478.

3. Goedert, R., Becker, R., Clements, A., and Whittaker, III, T. Time–resolved shadowgraphic imaging of the response of dilute suspension to laser pulses. *Journal of the Optical Society of America B 15*, 5 (1998), 1442–1462.

4. Jackson, J. D. *Classical Electrodynamics*. John Wiley & Sons, Inc., 1962.

5. Kerker, M. *The Scattering of Light and Other Electromagnetic Radiation*, vol. 16 of *Physical Chemistry–A Series of Monographs*. Academic Press, New York, 1969.

6. Lochte-Holtgreven, W., Ed. *Plasma Diagnostics*. North-Holland Publishing Company, 1968.

7. Mikic, B. B., Rohsenow, W. M., and Griffith, P. On bubble growth rates. *International Journal of Heat Mass Transfer 13* (1970), 657–665.

8. Reif, F. *Fundamentals of Statistical and Thermal Physics*. McGraw–Hill Book Company, 1965.

9. Stralen, S. J. D. V. The growth rate of vapour bubbles in superheated pure liquids and binary mixtures. *International Journal of Heat Mass Transfer 11* (1968), 1467–1489.

10. Stratton, J. A. *Electromagnetic Theory*, first ed. McGraw-Hill Book Company, Inc., 1941.

11. van de Hulst, H. *Light Scattering by Small Particles*. Dover Publications, Inc., 1981.

12. Witze, C. P., Schrock, V. E., and Chambre, P. L. Flow about a growing sphere in contact with a plane surface. *International Journal of Heat Mass Transfer 11* (1968), 1637–1651.

Vita

Mr. Gregg T. Anderson was born 14 April 1951 in Grafton, North Dakota. He graduated from St. Thomas Public School in St. Thomas, North Dakota in June 1969. He entered undergraduate studies at Mayville State University, Mayville, North Dakota, emerging with a Bachelor of Science degree in Education in August 1975. He did some graduate work in chemistry and then taught high school chemistry, mathematics and English within the school system of North Dakota. He later joined the Air Force and received a commission through the Officer Training School, Lackland AFB, TX in December 1981. He then spent the next two and a half years as a Second Lieutenant at Arizona State University where he earned a Bachelor of Science degree in Electrical Engineering in May 1984.

His first assignment was at Wright–Patterson AFB where he worked within the F–16 System Program Office on the LANTIRN program. After two years, he changed his assignment and worked for the B–1 System Program Office. In August 1987 he separated from the Air Force and joined the staff of the Materials and Manufacturing Directorate, Hardened Materials Branch as an electronics engineer. In September 1991, he entered the Applied Mathematics Program, Department of Mathematics and Statistics, Air Force Institute of Technology.

Permanent address: 6818 Packingham Dr.
Englewood, OH 45322

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dec 99	Master's Thesis

4. TITLE AND SUBTITLE
A Numerical Simulation of a Carbon Black Suspension Cell via a Time-Reversed, Double Layer Compute Algorithm

5. FUNDING NUMBERS

6. AUTHOR(S)
Gregg T. Anderson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2750 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAM/ENC/99D-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Maj. William D. Cowan
Materials and Manufacturing Directorate, Hardened Materials Branch
AFRL/MLPJ
3005 P. Street, Bldg. 651, Rm 169
Wright-Patterson AFB OH 45433-7702 ; ph (937) 255-3808 x 3148

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Advisor: Prof W. P. Baker, ENC
PH: 255-3636 X 4517
EMAIL: william.baker@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A numerical simulation of a carbon black suspension cell is explored which models a laser-induced plasma within a liquid ethanol medium of approximately 1 mm thickness. The simulation model assumes a laser pulse with a pulse width of approximately 9 nsecs propagating in the left-to-right direction striking the front surface of the medium and focusing to a spot within the liquid volume. When the energy density within a given irradiated volume is sufficiently high, it ignites the carbon particles and generates a large number of free electrons, i.e. a plasma. The plasma couples with the in-coming laser energy on a picosecond timescale, thereby attenuating the intensty of the remaining pulse as it traverses the medium. The simulation divides the sample into descrete layers and models the propagation of the plasma through the course of a single laser pulse containing a total fluence of 1 Joule/cm^2. A new double layer, time-reversed algorithm is employed which modifies and extends the capabilities of the existing code. The older version is used as a baseline for comparison with the new program.

14. SUBJECT TERMS	15. NUMBER OF PAGES
carbon black suspension, simulation, numerical modeling, plasma, algorithm, layer	98
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL